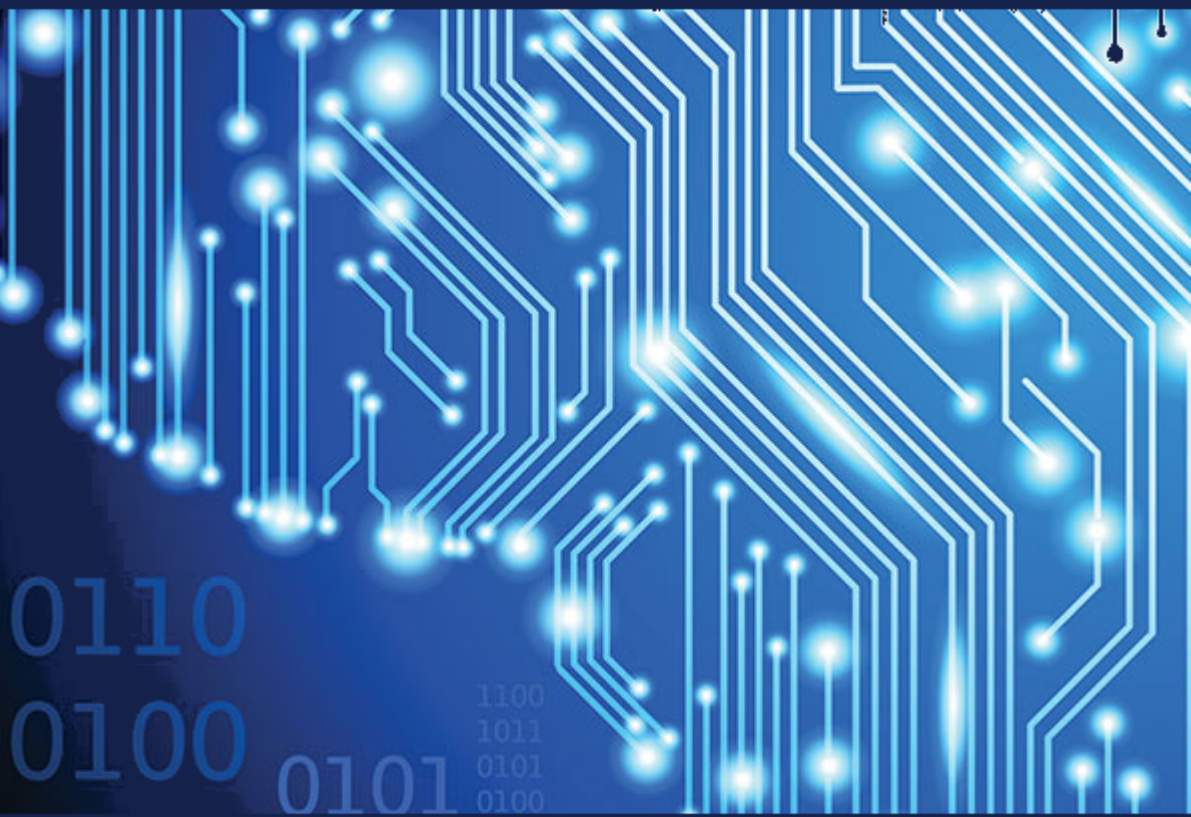
Воронина В.В., Михеев А.В.

Теория и практика машинного обучения



УлГТУ 2017

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение
высшего образования
«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В. В. Воронина, А. В. Михеев,

Н. Г. Ярушкина, К. В. Святов

ТЕОРИЯ И ПРАКТИКА МАШИННОГО ОБУЧЕНИЯ

Учебное пособие

Ульяновск
УлГТУ
2017

УДК 004.8 (075)
ББК 32.973-018.1я7
В12

Рецензенты:

Главный научный сотрудник ФНЦП АО «НПО «Марс», д-р техн. наук Токмаков Г. П.

Начальник расчетно-теоретического отдела ОАО «Ульяновское конструкторское бюро приборостроения», канд. техн. наук Сорокин М. Ю.

Утверждено редакционно-издательским советом университета в качестве учебного пособия

Воронина, Валерия Вадимовна

В12 Теория и практика машинного обучения : учебное пособие / В. В. Воронина, А. В. Михеев, Н. Г. Ярушкина, К. В. Святов. – Ульяновск : УлГТУ, 2017. – 290 с.

ISBN 978-5-9795-1712-4

Учебное пособие рассматривает вопросы, связанные с анализом данных: модели, алгоритмы, методы и их реализацию на языке Python. Особое внимание уделено анализу временных рядов. Книга предназначена для студентов группы направлений 09, а также для студентов других групп направлений, изучающих дисциплины, связанные с разработкой приложений в области анализа данных, в том числе TimeSeriesDataMinig и DataMining.

УДК 004.8 (075)
ББК 32.973.2-018.2я7

ISBN 978-5-9795-1712-4

Содержание

Введение

На протяжении всего периода своего развития человечество постоянно ищет способы автоматизации тех или иных видов деятельности, в итоге перекладывая на «плечи» механизмов все более обширные и сложные обязанности. Изначально мечты о роботах, выполняющих всю работу, находили воплощение лишь в литературных произведениях, однако с развитием технологий им открылся путь и в реальность. Сейчас уже не столь фантастичными звучат слова песни «До чего дошел прогресс» (к/ф «Приключения Электроника»): «До чего дошел прогресс – труд физический исчез, да и умственный заменит механический процесс». Ведь эти строки достаточно полно отражают суть такой дисциплины, как «машинное обучение», благодаря которой их воплощение в жизнь становится возможным. Однако стоит заметить, что до полной реализации описанного в песне еще очень далеко: пока машинное обучение не способно заменить умственный труд, и в ближайшее время вряд ли сможет. Но шаги в этом направлении активно совершаются.

С теоретической стороны машинное обучение – дисциплина, находящаяся на пересечении математической статистики, численных методов оптимизации, теории вероятностей, а также дискретного анализа. С помощью ее методов происходит решение задачи извлечения знаний из данных, которой занимается еще только формирующаяся область «Интеллектуальный анализ данных» (DataMining). Согласно Википедии [1]: «В теории искусственного интеллекта и экспертных систем, знание – это совокупность утверждений о мире, свойствах объектов, закономерностях процессов и явлений, а также правил логического вывода одних утверждений из других и правил использования их для принятия решений. Главное отличие знаний от данных состоит в их структурности и активности: появление в базе знаний новых фактов или установление новых связей между ними может стать источником изменений в принятии решений». Если,

опираясь на это определение знания, рассмотреть DataMining как процесс, то его сутью будет нахождение в «сырых» данных знаний, обладающих свойствами нетривиальности, интерпретируемости и практической полезности для принятия решений в различных сферах деятельности человека. Причем, эти знания ранее неизвестны.

С практической же стороны машинное обучение нацелено на создание систем, способных адаптироваться к решению различных задач без явного кодирования алгоритма, то есть систем, способных обучаться.

В данной книге рассмотрены основные модели и алгоритмы машинного обучения, в том числе для анализа временных рядов. Наряду с этим представлены и их практические реализации на языке Python. В последних разделах книги обучающемуся предлагаются контрольные вопросы по пройденным темам, а также задачи для выполнения, с помощью которых он сможет проверить себя и закрепить полученные навыки.

Книга подготовлена при реализации проекта №2.4760.2017/8.9 «Исследование и разработка моделей, методов и алгоритмов гибридизации нечетких предметных онтологий, логического вывода и интеллектуального анализа временных рядов» в рамках государственного задания Минобрнауки Российской Федерации.

О ЗАДАЧАХ МАШИННОГО ОБУЧЕНИЯ

Как было сказано выше, машинное обучение нацелено на создание систем, способных получать знания из данных, а также способных с помощью обучения улучшать показатели своей работы. Таким образом, можно утверждать, что машинное обучение является одной из областей науки о данных (DataScience). На рисунке 1 представлены ее составляющие и показано место машинного обучения среди них.

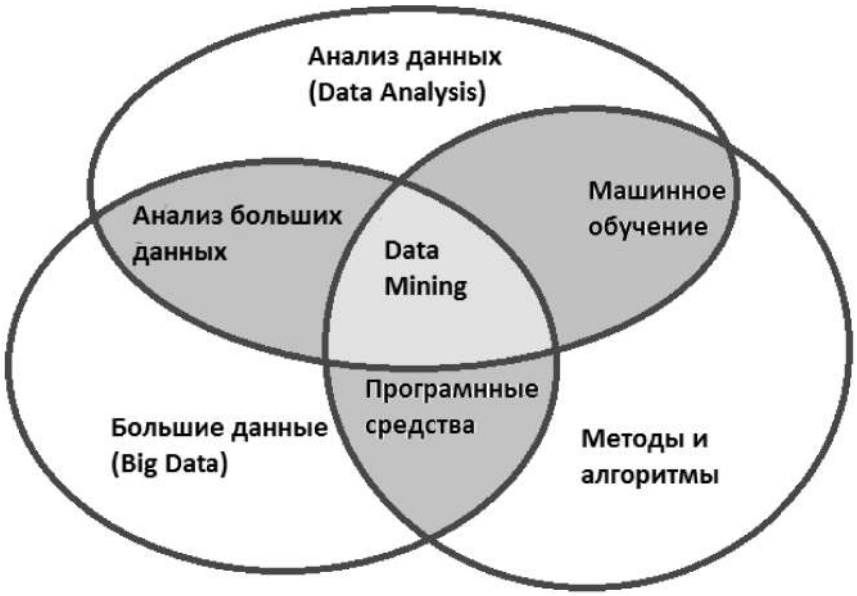


Рисунок 1. Области науки о данных

Сложность работы в сфере DataMining обуславливается неточностью, противоречивостью, разнородностью, неполнотой данных, которые при этом могут иметь гигантские объемы. Для их обработки требуются специальные программные средства: инструменты преобразования «сырых» данных в информацию, а информацию в знания. Кроме того, алгоритмы обработки данных должны иметь возможность обучаться по прецедентам. Машинное обучение (МО) как раз и занимается разрешением подобных сложностей. Причем в настоящее время усилия сконцентрированы на повышении

не столько качества решения, сколько скорости и оптимальности его получения.

Как видно из рисунка 1, машинное обучение пересекается с анализом данных, то есть имеет общие черты, но также и свою специфику. Анализ данных – это огромная и уже относительно не молодая область математики и информатики. Под нее попадает всяческая обработка данных. Например, обработка данных физического эксперимента статистическими методами для получения обобщенной информации (усредненных значений, точных значений, доверительных интервалов, оценок и тому подобное), обработка сигнала методом Фурье, даже подсчет минимального и максимального значений или отношения каких-то величин может быть анализом данных, если эти операции влекут за собой важные выводы, знания о данных и позволяют решить поставленную задачу. Так получение дисперсии некой величины является анализом данных, но не является в чистом виде машинным обучением. Хотя важно отметить, что методы статистики имеют большое значение и в машинном обучении, о чем мы еще будем говорить. Отличительной особенностью методов машинного обучения является то, что они способны решать широкий спектр задач без явного кодирования алгоритма решения. Например, метод Регрессии входит в машинное обучение. Регрессия давно применяется в экономике для прогнозирования и оценки разнообразных параметров, потому что данная математическая модель в некотором роде универсальна. Однако стоит заметить, что регрессия появилась в математике гораздо раньше, чем ее стали использовать в экономике, а вот машинным обучением это стало только тогда, когда все эти методы начали рассматриваться не как обособленные механизмы решения задач из специальных областей, а как методы преобразования информации вообще.

На рисунке 2 представлена диаграмма распределения задач, которые приходится решать в сфере машинного обучения. Необходимо отметить, что соотношение областей на данной диаграмме

отражает лишь субъективное видение отрасли авторами на момент написания книги.

Рисунок 2. Типы задач машинного обучения

Понятно, что разработка программ в области машинного обучения весьма дорога. Поэтому применять его следует лишь в случаях, когда это действительно необходимо. В таблице 1 представлены некоторые признаки, по которым можно принять решении о необходимости применения к задаче методов машинного обучения.

Таблица 1. Ключевые характеристики решаемых задач

Целесообразно применение классических методов	**Целесообразно применение методов машинного обучения**
Наличие четко формализованного алгоритма (например, поиск клиентов, которые тратят больше всего денежных средств в месяц или чаще всего покупают. Так называемая RFM сегментация).	Отсутствие четко формализованного алгоритма решения ввиду высокой вариативности решения (например, распознавание рукописного текста).
Не допускается неопределенность поведения модели (например, расчет местоположения самолета по строго заданным формулам).	Допускается неопределенность поведения модели (например, предварительный контроль качества детали, где допускается определенная доля брака).

Окончание таблицы 1

Целесообразно применение классических методов	Целесообразно применение методов машинного обучения
Экономическая целесообразность (например, методы МО дают 98% качества, классическое решение дает 90% качества, что является приемлемым, но решение на основе МО в 2 раза дороже и реализуется дольше). Поэтому здесь выбирается приоритетный критерий.	

Кроме того, для разработки решений на основе МО необходимы оцифрованные данные. Они являются ключевым аспектом, от которого зависит качество и полнота решения. Таким образом, очень важно понимать, что для применения методов машинного обучения требуются собранные и специально подготовленные данные.

Как правило, процесс машинного обучения проходит несколько этапов. На рисунке 3 представлен обобщенный алгоритм, выполняющийся при решении задач методами рассматриваемой области.

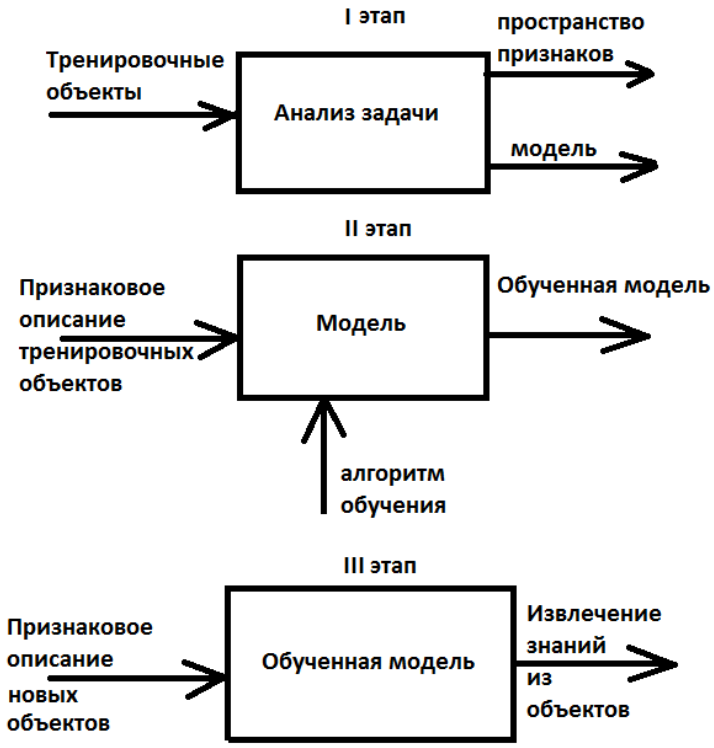


Рисунок 3. Этапы машинного обучения

Необходимо отметить, что данная иллюстрация отражает идеальный случай, когда после обучения мы сразу получили работающую модель. Однако в реальности, как правило, между вторым и третьим этапом есть промежуточное, но очень важное действие – оценка качества модели. И именно по его результатам принимается решение о переходе на следующий этап или о возврате к одному из предыдущих. Этот вопрос мы рассмотрим подробнее далее, здесь же введем определение пространства признаков, для лучшего понимания схемы с рисунка 3.

Пространство признаков

Пространство признаков – это N-мерное пространство, где N – число измеряемых характеристик объектов, выделенное для конкретной задачи. При этом объекты в пространстве признаков задаются N-мерными векторами, каждая компонента которых представляет собой значение определенной характеристики.

Рассмотрим пример. Пусть необходимо разработать функцию для сайта салона красоты, которая будет предлагать клиентам определенные косметические процедуры. Понятно, что они будут разными для разных возрастных и гендерных групп. То есть в этой задаче ключевыми будут характеристики пола и возраста людей. Графическое отображение пространства признаков, а также заданные в нем объекты A и B показаны на рисунке 4. В векторном виде объект A с характеристиками «пол» = «женский» и «возраст» = 40 будет задан как: $A= \{40, ж\}$. Объект B с возрастом 50 и мужским полом – $B= \{50, м\}$.

Как видно из рассмотренного примера, признаки объектов могут быть разными по своей природе. Существует следующая типизация:

1. Бинарные: $F \epsilon \{0,1\}$.
2. Номинальные: $|F| < \infty$.
3. Номинальные, упорядоченные: $|F| < \infty$, $F_i < F_{i+1} \ldots < F_{i+n}$.
4. Количественные: $F \epsilon R$.

Каждый тип признаков обрабатывается по-разному, о чем скажем далее. Пока же продолжим разговор о других аспектах рассматриваемой дисциплины. Как неоднократно было сказано, работа в сфере машинного обучения нацелена на создание систем, способных обучаться. Рассмотрим формальное определение понятия «обучение».

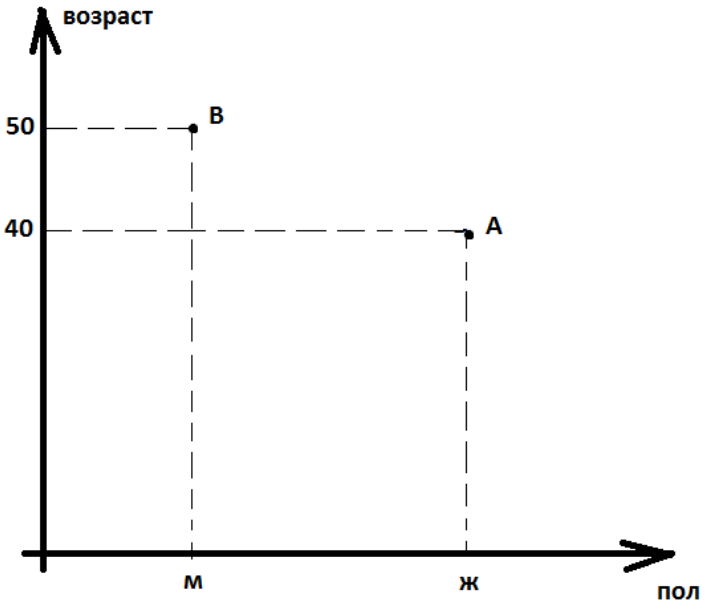


Рисунок 4. Объекты в пространстве признаков

Формальное определение понятия «обучение»

Формально понятие «обучение» специалисты определяют так: «Пусть есть некое множество задач класса *C,* для которого задано некоторое множество опыта *EX* и определена мера качества *L*. Тогда о наличии обучения на опыте *EX* относительно класса задач *C* в смысле меры качества *L* можно говорить в том случае, если при предъявлении нового опыта *EX'* возрастает качество решения задачи класса *C,* измеряемое мерой *L»*.

Существуют два основных типа обучения: с учителем и без учителя. Более подробно о них будет рассказано далее, здесь же рассмотрим формальную постановку задачи обучения для самого первого (и самого простого) типа – обучения с учителем.

Пусть X – некоторое множество объектов, по которым необходимо получить некоторое множество ответов Y. При этом предполагается, что существует некоторая зависимость $y: X \to Y$. Тогда задача обучения будет формулироваться так: дано $\{x_1 \dots x_n\} \in X$ – обучающая выборка, и $\{y_1 \dots y_n\}$ – известные ответы, причем $y_i = y(x_i)$. Найти: $a: X \to Y$ – решающую функцию, приближающую y на всем множестве X.

Данная задача в свою очередь порождает ряд подзадач:

1. Определение способа задания объектов;
2. Определение способа задания ответов;
3. Определение способа построения функции a;
4. Определение способа приближения для a и y.

Все эти подзадачи решаются последовательно, и результаты решения предыдущей, как правило, влияют на решение текущей. Поэтому самой важной среди них будет определение способа описания данных. Можно выделить следующие типы входных данных при обучении:

1. Координаты объектов в пространстве признаков;
2. Временной ряд;
3. Сигнал;
4. Изображение;
5. Видеоряд;
6. Описание взаимоотношений между объектами.

На основе описанного выше введем формальное описание приведенных на рисунке 3 второго и третьего этапов машинного обучения.

Пусть $X = \{x_1 \dots x_k\}$ – исследуемые объекты. $F = \{f_1 \dots f_N\}$ – пространство признаков, в котором эти объекты будут рассматриваться. При этом $f_i(x_j)$, $i \in [1 \dots N]$, $j \in [1 \dots k]$ – значение i-го признака j-го объекта. Тогда признаковое описание тренировочных объектов задается в виде матрицы

$$D = || f_i(x_j) ||_{k \times N} = \begin{pmatrix} f_1(x_1) \dots & f_N(x_1) \\ f_1(x_k) \dots & f_N(x_k) \end{pmatrix}.$$

Этап обучения может быть формализован следующим образом. Метод обучения $\mu: (X \times Y)_k \rightarrow A$ по выборке $XY_k=(x_i,y_i)$, $i \in [1 \ldots k]$ строит алгоритм $a=\mu(XY_k)$:

$$\begin{pmatrix} f_1(x_1) \ldots & f_N(x_1) \\ f_1(x_k) \ldots & f_N(x_k) \end{pmatrix} \xrightarrow{y} \begin{pmatrix} y_1 \\ y_k \end{pmatrix} \xrightarrow{\mu} a$$

Этап применения обученной модели формализуется следующим образом. Алгоритм a для новых объектов $X'=\{x'_1 \ldots x'_k\}$ выдает ответы $y_i'=a(x'_i)$, $i \in [1 \ldots k]$:

$$\begin{pmatrix} f_1(x'_1) \ldots & f_N(x'_1) \\ f_1(x'_k) \ldots & f_N(x'_k) \end{pmatrix} \xrightarrow{a} \begin{pmatrix} y'_1 \\ y'_k \end{pmatrix}.$$

Общий алгоритм машинного обучения

С понятием обучения тесно связано понятие обобщающей способности. Обобщающая способность – это свойство модели отражать исходные данные в требуемые результаты ($X \rightarrow Y$) на всем множестве исходных данных (во всех сценариях, а не только на тренировочных примерах). Величину обобщения оценивают через обратную величину – отклонение или ошибку. Ошибка – это численно выраженная разница между ответом модели и требуемым (реальным) значением. В более общем смысле обобщающая способность – способность модели найти некий «закон природы», который будет описывать неизвестную нам и скрытую взаимосвязь входных и выходных данных. Таким образом, опираясь на понятие обобщающей способности, можно выделить основные проблемные вопросы машинного обучения:

1. Достаточно ли данных для нахождения в них полезных знаний?
2. Может ли в принципе данная модель обучиться на имеющихся данных?
3. Будет ли полученная модель приближать требуемый «закон природы» на всем возможном множестве X?

4. Насколько хорошо будет работать обученная модель или насколько часто и насколько сильно будет ошибаться модель на реальных (контрольных) данных?

По сути в машинном обучении решается задача приближения алгоритма к некоторому «закону природы». С математической точки зрения задача приближения является задачей оптимизации или минимизации ошибки:

$$E=|y-y'|\rightarrow min\ .$$

Таким образом, общий алгоритм решения задач в сфере машинного обучения будет состоять из следующих шагов:

1. Понимание задачи и исходных данных;
2. Формулировка решения на математическом языке (на этом шаге важно понять, что задача формализуема, а результаты работы модели могут быть проверены);
3. Предобработка данных и выделение ключевых признаков;
4. Построение модели (или моделей);
5. Обучение модели (моделей) и оценка качества;
6. Эксплуатация модели при достижении требуемого качества, либо возврат к одному из предыдущих шагов (перенастройка модели, добыча новых данных и т. п.).

Типы задач машинного обучения

Машинное обучение применяется для решения задач в следующих областях (основные сферы применения):

1. Медицинская диагностика.
2. Техника, в частности:
    2.1. Автоматизация и управление.
    2.2. Техническая диагностика.
    2.3. Робототехника.
    2.4. Компьютерное зрение.
    2.5. Распознавание речи.
3. Экономика, в частности:
    3.1. Кредитный скоринг (credit scoring).

3.2. Предсказание ухода клиентов (churm prediction).

3.3. Обнаружение мошенничества (fraud detection).

3.4. Биржевой технический анализ (technical analysis).

3.5. Биржевой надзор (market surveillance).

4. Офисная автоматизация, в частности:

4.1. Распознавание текста.

4.2. Обнаружение спама.

4.3. Категоризация документов.

4.4. Распознавание рукописного ввода.

Если же говорить об обобщенных типах задач машинного обучения, то можно выделить следующие:

1. Регрессия (или иногда встречается термин «аппроксимация»);
2. Классификация;
3. Кластеризация.

Помимо указанных видов существуют и другие, но остановимся на обозначенных, так как они наиболее распространены. Рассмотрим формальную постановку этих задач.

Задача регрессии – приближение неизвестной целевой зависимости на некотором множестве данных. Пусть X – множество данных – описаний некоторых объектов. Y – множество возможных ответов для X.

В задаче регрессии предполагается, что существует неизвестная целевая зависимость $y: X \to Y$, чьи значения известны только на объектах обучающей выборки $XY=\{(x_1,y_1)\dots (x_n,y_n)\}$, $x \in X$, $y \in Y$. Необходимо получить алгоритм $a: X \to Y$, приближающий целевую зависимость как на множестве XY, так и на X. То есть решить задачу регрессии – значит найти алгоритм, обладающий способностью к обобщению эмпирических фактов (способностью к выводу общих знаний из частных наблюдений, прецедентов).

Задача классификации – распределение некоторого множества объектов по заданному множеству групп (классов). При этом есть некоторое подмножество объектов, для которых распределение по классам известно, классовая принадлежность остальных – неизвестна.

Требуется построить алгоритм, который указывал бы классовую принадлежность для любого объекта из исходного множества.

Формально постановку задачи классификации можно описать следующим образом. Пусть X – множество данных – описаний некоторых объектов. Y – конечное множество классов, отмеченных метками. Существует неизвестная целевая зависимость – отображение $y: X \to Y$, чьи значения известны только на объектах обучающей выборки $XY = \{(x_1, y_1) \dots (x_n, y_n)\}$, $x \in X$, $y \in Y$. Необходимо получить алгоритм $a: X \to Y$, способный классифицировать произвольный объект $x \in X$.

Как можно заметить, данная задача схожа с предыдущей. Однако главная особенность задачи регрессии заключается в том, что функция $a: X \to Y$ является непрерывной вещественной функцией. Задача классификации отличается от этого тем, что Y – дискретное множество. Кроме того, в отличие от задачи аппроксимации у задачи классификации выделяют несколько типов. По количеству классов можно выделить:

1. Классификацию на два класса: множество Y содержит всего две метки.
2. Классификацию на множество классов: Y содержит от трех до нескольких тысяч меток.

По характеру разделения объектов на классы можно выделить:

1. Классификацию на непересекающиеся классы: один объект принадлежит только одному классу.
2. Классификацию на пересекающиеся классы: один объект может принадлежать нескольким классам.
3. Классификацию на нечеткие множества: объект принадлежит всем классам с определенной степенью принадлежности.

Задача кластеризации – разделение некоторого множества объектов на непересекающиеся группы (кластеры) таким образом, чтобы каждая группа состояла из схожих объектов, а объекты разных кластеров существенно отличались.

Формально постановку задачи кластеризации можно описать следующим образом. Пусть X – множество данных – описаний некоторых объектов. Y – множество кластеров, отмеченных метками. Определена функция расстояния между объектами из исходного множества X: $f(x, x')$, и есть некоторая обучающая выборка объектов $X_o = \{x_1 \dots x_n\}$, $x \in X$. Необходимо разбить обучающую выборку на кластеры, приписав каждому x номер кластера y_i, так, чтобы близкие по метрике f объекты принадлежали одному кластеру, а объекты разных кластеров существенно отличались по метрике f. То есть необходимо построить алгоритм $a: X \rightarrow Y$, который любому $x \in X$ ставит в соответствие номер кластера $y \in Y$. Причем, иногда множество Y известно заранее, но чаще все-таки ставится задача получить оптимальное число кластеров, исходя из характера данных. Оптимальность оценивается по какому-либо критерию качества кластеризации.

Задача кластеризации сложнее аппроксимации и классификации. Это обусловлено следующими причинами:

1. Нет однозначного критерия качества кластеризации. Существует ряд эвристических критериев, а также ряд бескритериальных алгоритмов, выполняющих вполне осмысленную кластеризацию, но дающих на одних и тех же данных разные результаты.
2. Число кластеров, как правило, заранее неизвестно и задается субъективно.
3. На результат кластеризации существенное влияние оказывает выбранная метрика расстояния, которая, как правило, также выбирается субъективно.

Однако, несмотря на описанные выше сложности, кластеризация помогает достичь следующие цели:

1. Улучшить понимание данных за счет выявления их кластерной структуры: разбиение объемной выборки на группы схожих объектов может упростить дальнейшую обработку

данных за счет применения к каждому кластеру своих методов анализа.

2. Осуществить сжатие данных. В данном случае подразумевается сокращение объемной выборки за счет работы с наиболее яркими представителями кластеров (групп схожих объектов).
3. Выявить новизну в массиве данных: обнаружить нетипичные объекты, которые не удается отнести ни к одному кластеру.
4. Решить задачу таксономии: построить древообразную иерархическую структуру, упорядочивающую исходные данные. Ее построение достигается за счет дробления крупных кластеров на более мелкие, которые в свою очередь также дробятся на еще более мелкие. Визуально таксономия отображается в виде графика – дендрограммы.

Способы обучения и оценки его качества

Как было сказано выше, основная характеристика систем, разрабатываемых с помощью методов машинного обучения, – способность к обучению. В зависимости от видов решаемых задач применяют различные алгоритмы реализации этой ключевой особенности. В рамках данного пособия рассмотрим три основных вида обучения, а также определим классы задач, подходящие для каждого из этих видов.

Первый тип – обучение с учителем. Формально он был описан ранее, поэтому здесь не будем на этом останавливаться подробно. Вкратце же обучение с учителем можно описать следующим образом: дано некоторое множество объектов и множество возможных реакций системы на эти объекты. При этом ответы и объекты связаны между собой некоторой неизвестной зависимостью. Есть конечная совокупность пар объект-ответ (прецедентов), называемая обучающей выборкой. На ее основе необходимо выявить алгоритм, который впоследствии для любого объекта из исходного множества даст достаточно точный ответ. Для измерения точности ответов используется один

из функционалов качества, как правило, завязанный на вычислении отклонения полученного ответа от ожидаемого, то есть вычислении ошибки. Рассмотрим некоторые их виды. В приведенных ниже формулах используются следующие обозначения: $XY=\{(x_1,y_1)\dots(x_n,y_n)\}$ – обучающая выборка, n – количество прецедентов, y_i – фактическое значение (ожидаемый ответ) в *i-м* прецеденте ($y_i \in XY$), $\hat{y}_i$ – выданный системой ответ для x_i ($x_i \in XY$).

1. Средняя ошибка представляет собой усреднение ошибок для каждого образца и вычисляется по формуле

$$\text{СО} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)}{n}.$$

2. Средняя абсолютная ошибка представляет собой усреднение абсолютных ошибок на каждом шаге и вычисляется по формуле

$$\text{САО} = \frac{\sum_{i=1}^{n}|y_i - \hat{y}_i|}{n}.$$

3. Среднеквадратическая ошибка вычисляется как сумма средних квадратов ошибок. Формула:

$$\text{СКО} = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}.$$

4. Корень из среднеквадратической ошибки вычисляется по формуле

$$\text{КСКО} = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}.$$

5. Средняя относительная ошибка вычисляется как среднее относительных ошибок:

$$\mathrm{CO} = \frac{\sum_{i=1}^{n} \frac{(y_i - \hat{y}_i)}{y_i}}{n} \times 100\%.$$

6. Средняя абсолютная относительная ошибка вычисляется как среднее относительных ошибок по модулю:

$$\mathbf{MAPE} = \frac{\sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right|}{n} \times 100\% .$$

7. Симметричная средняя абсолютная относительная ошибка вычисляется как

$$\mathbf{SMAPE} = \frac{\sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{(y_i + \hat{y}_i)/2} \right|}{n} \times 100\% .$$

У всех представленных мер качества есть свои достоинства и недостатки. Например, у первой и пятой недостаток заключается в том, что положительные и отрицательные ошибки аннулируют друг друга, поэтому в некоторых случаях они не являются достаточно хорошими индикаторами качества. В связи с этим чаще всего используется третья или четвертая меры.

Обучение с учителем используется при решении задач аппроксимации и классификации. В первом случае ответы являются действительными числами или векторами, во втором – выбираются из конечного множества меток-классов. Необходимо отметить, что приведенные выше формулы подходят только для случаев, когда ответ системы и требуемый ответ – действительные числа, получаемые при решении задачи аппроксимации. В задачах классификации же оценка качества чаще всего завязана на соотношении количеств правильно и неправильно отнесенных к классам объектов.

Второй тип обучения – обучение без учителя. Формально постановку задачи обучения без учителя можно описать следующим образом. Пусть X – множество данных – описаний некоторых

объектов. Необходимо найти множество Y, состоящее из взаимосвязей $f: (x, x')$ между объектами из X $(x, x' \in X, f \in Y)$. Качество выявления взаимосвязей проверяется некоторой метрикой, выбранной исходя из решаемой задачи.

Обучение без учителя используется для решения следующих типов задач:

1. Задача кластеризации.
2. Поиск правил ассоциации.
3. Сокращение размерности данных.
4. Визуализация данных.

В определенной степени каждая из последних трех задач является производной от первой или ее частным случаем. Рассмотрим подробнее формулировки названных задач.

Под задачей поиска правил ассоциаций подразумевается выявление в признаковых описаниях объектов (исходных данных) таких наборов и значений признаков, которые особенно часто (неслучайно часто) встречаются в исходных данных. Если же проводить аналогию с первой задачей, то каждое правило в данном случае может быть представлено как кластер.

Задача сокращения размерности данных состоит в следующем. Существует большой (значительно большой) объем признаковых описаний объектов. Причем этот объем обуславливается внушительным количеством измерений признакового пространства. Необходимо представить те же данные в пространстве меньшей размерности, при этом минимизировав потери информации. Группировка по кластерам как раз и будет одним из вариантов решения проблемы.

Задача визуализации данных является по сути частным случаем предыдущей: ее цель – представить исходные данные в отображаемом пространстве, то есть пространстве размерности 2 или 3.

Как следует из описанного выше, обучение без учителя в какой-то мере так или иначе сводится к кластеризации. Поэтому для оценки качества обучения данным способом, как правило, используют метрики качества кластеризации. Причем при их выборе учитывается,

что эти метрики не должны зависеть от исходных данных, а только от результатов разбиения. Все оценки качества можно разделить на внешние и внутренние. Первые используют внешнюю информацию об истинном разбиении объектов на кластеры, вторые опираются только на набор исходных данных, то есть данные метрики могут работать с неразмеченной выборкой, когда заранее не известно истинное разбиение объектов на группы. И именно с их помощью определяют оптимальное число кластеров.

Приведем в качестве примера метрики, выделенные компанией ODS [2]:

1. Adjusted RandIndex (*ARI*). Данная метрика относится к группе внешних: предполагается, что истинные метки объектов известны (например, заданы экспертом), однако от самих значений меток она не зависит. Только от разбиения выборки на кластеры.

Рассчитывается мера следующим образом: пусть n – число объектов в выборке, a – число пар объектов, имеющих одинаковые метки и находящихся в одном кластере, b – число пар объектов, имеющих различные метки и находящиеся в различных кластерах. Тогда можно посчитать долю объектов, для которых исходное и полученное в результате кластеризации разбиения согласованы:

$$\boldsymbol{RI} = \frac{\boldsymbol{2(a+b)}}{\boldsymbol{n(n-1)}} .$$

Полученная величина называется RandIndex (*RI*) и выражает схожесть двух разных кластеризаций одной и той же выборки. Чтобы этот индекс давал значения, близкие к нулю, для случайных кластеризаций при любом n и числе кластеров, необходимо произвести его нормирование, то есть получить AdjustedRandIndex:

$$\boldsymbol{ARI} = \frac{\mathbf{RI} - \mathbf{E[RI]}}{\boldsymbol{max}(\mathbf{RI}) - \mathbf{E[RI]}} ,$$

где E – математическое ожидание.

Мера *ARI* симметрична и не зависит от перестановок и значений меток. По сути этот индекс является мерой расстояний между различными разбиениями выборки. Его область значений – [-1,1]. Интерпретировать ее интервалы можно следующим образом: для «независимых» разбиений на кластеры – отрицательные значения, для случайных разбиений – близкие к нулю, для схожих разбиений – положительные значения, причем, *ARI*=1 говорит о совпадении разбиений.

2. Adjusted MutualInformation (*AMI*). Данная метрика схожа с предыдущей: она также симметрична и не зависит от значений и перестановок меток. Для ее определения используется функция энтропии. Разбиения выборки интерпретируются как дискретные вероятности: вероятность отнесения к кластеру равна доле объектов в нем. Как и в предыдущем случае, для данной метрики необходимо вычислить специальный индекс – MutualInformation (*MI*). Данная величина определяется как взаимная информация для двух распределений, соответствующих разбиениям выборки на кластеры. Интерпретировать это можно как долю информации, общей для обоих разбиений: насколько информация об одном из них уменьшает неопределенность относительно другого. Этот индекс рассчитывается по следующей формуле:

$$\boldsymbol{MI(X,Y)} = \sum_{\boldsymbol{y}\in\boldsymbol{Y}}\sum_{\boldsymbol{x}\in\boldsymbol{X}}\boldsymbol{p(xy)}\log\left(\frac{\boldsymbol{p(xy)}}{\boldsymbol{p(x)p(y)}}\right),$$

где *p(x, y)* – совместная функция распределения вероятностей для *X* и *Y; p(x)* и *p(y)* – функции распределения предельной вероятности для *X* и *Y* соответственно.

Областью значения индекса *AMI* является диапазон [0,1]. Интерпретируется он следующим образом: значения, близкие к нулю, говорят о независимости разбиений, а близкие к единице – об их схожести (или совпадении при *AMI*=1).

3. Гомогенность, полнота, V-мера. Данные метрики рассматривают разбиение выборки как дискретные распределения и определяются с использованием функции энтропии и условной энтропии.

Гомогенность определяет, насколько каждый кластер состоит из объектов одного класса, и рассчитывается по формуле

$$h = 1 - \frac{H(C|K)}{H(C)},$$

где K – результат кластеризации, C – истинное разбиение, H – функция энтропии. При этом

$$\boldsymbol{H(X) = -\sum_{i=1}^{n} P(x_i)\log_b P(x_i)},$$

$$\boldsymbol{H(X|Y) = \sum_{j} p(x_i y_j)\log \frac{p(y_j)}{p(x_i y_j)}}.$$

Полнота измеряет, насколько объекты одного класса относятся к одному кластеру и определяется по формуле

$$c = 1 - \frac{H(K|C)}{H(K)}.$$

Эти меры принимают значения в диапазоне [0,1]. Интерпретировать их можно следующим образом: чем больше значение, тем более точна кластеризация. Эти меры не являются симметричными и не являются нормализованными, в отличие от рассмотренных ранее, и поэтому они зависят от числа кластеров. Согласно ресурсу [2]: «Случайная кластеризация не будет давать нулевые показатели при большом числе классов и малом числе объектов. В этих случаях предпочтительнее использовать *ARI*. Однако при числе объектов более 1000 и числе кластеров менее 10 данная проблема не так явно выражена и может быть проигнорирована».

Чтобы учесть значения обеих величин, вводится симметричная *V*-мера, показывающая, насколько кластеризации схожи между собой. Ее расчет происходит по формуле

$$v = 2\frac{hc}{h+c}.$$

Силуэт. Данная метрика относится к типу внутренних: она не предполагает знания об истинном разбиении и первоначально рассчитывается для каждого объекта следующим образом. Пусть a – среднее расстояние от данного объекта до объектов из того же кластера,

b – среднее расстояние от данного объекта до объектов из ближайшего кластера (но не того, в котором находится сам объект). Тогда силуэтом данного объекта будет величина, вычисляемая по формуле

$$s = \frac{b - a}{\max(ab)}.$$

Силуэтом же всей выборки будет средняя величина силуэтов ее объектов. Интерпретировать метрику можно следующим образом: она показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров. Область значения данной величины – диапазон [-1,1]. При этом значения, близкие к левой границе, соответствуют плохим (разрозненным) кластеризациям; значения, близкие к середине, говорят о пересечении и наложении кластеров; значения, близкие к правой границе, соответствуют «плотным», четко выделенным кластерам. Согласно [2]: «чем больше силуэт, тем более четко выделены кластеры, и они представляют собой компактные, плотно сгруппированные облака точек».

Данная величина может быть использована для выбора оптимального числа кластеров, если оно заранее неизвестно: оно будет равно числу, максимизирующему значение силуэта. В отличие от рассмотренных ранее метрик, данная величина зависит от формы кластеров. Силуэт достигает больших значений при более выпуклых кластерах, которые получаются при помощи алгоритмов, основанных на восстановлении плотности распределения.

Типовые задачи при подготовке данных и обучении моделей

В разделе «Общий алгоритм машинного обучения» были представлены шаги разработки решения. Если задача типовая и не требует специального исследования или разработки специального метода, то существенная работа начинается с п. 3, который посвящен подготовке данных (обучающей и тестовой выборке). Исходные данные в подавляющем большинстве случаев требуют предварительной обработки перед тем как на них будет обучаться модель. Как правило, предварительная обработка подразумевает следующую работу: учет

пропусков, кодирование нечисловых данных, приведение данных к единому масштабу и стандартизация данных, разметка данных. Рассмотрим их подробнее.

Учет пропусков

В реальных задачах данные могут содержать пропуски. Это может быть вызвано тем, что клиенты не заполняли все поля в анкете или аккаунте, или тем, что не все параметры были оцифрованы за все время работы системы.

Таким образом, встает вопрос о том, как интерпретировать пропуски. Простейший вариант заключается в исключении объектов, имеющих неполные сведения (то есть удаление тех строк из матрицы признаков, в столбцах которых есть пропуски), или исключении признаков, содержащих неполные сведения (то есть удаления тех столбцов, в которых есть пропуски). Плюсы этого варианта в том, что он очень прост и его можно сразу опробовать на какой-либо модели. Минусы вполне очевидны. Если много объектов будут иметь пропуски, то мы рискуем удалить важные аспекты закономерности, которая сокрыта в данных, и, как следствие, получим низкое качество аппроксимации. Если мы удалим столбец, который содержит крайне важный признак, мы рискуем вовсе потерять ключевую информацию о разделении объектов.

Второй вариант борьбы с пропусками заключается в том, чтобы заменить их при помощи интерполяции. Это может быть среднее или медианное значение по столбцу. В случае если признак является функцией от времени, как и отдельные объекты, то можно интерполировать пропуски только по соседним временным значениям (например, если объект представляет собой вероятность покупки квартиры, а признак – среднюю заработную плату программиста в этот год, то за пропущенный год цифру можно приблизить по двум соседним).

Третий вариант можно также рассмотреть на примере с квартирой, который был описан выше. Если речь идет о каких-то открытых

экономических, социальных или других параметрах, то их можно найти в других источниках и тем самым дополнить данные.

Четвертый вариант заключается в том, чтобы закодировать пропуски специальным числовым значением (это может быть число, не встречающееся у других объектов) или категориальным значением (представление категориальных признаков будет рассмотрено ниже). Такой подход, как минимум, позволит внести информацию о том, что эти объекты по этому признаку отличаются от других (то есть мы не сообщим точной информации, но не удалим объекты полностью, как в первом подходе).

Пятый подход заключается в привлечении эксперта в соответствующей теме, который сможет сказать, как именно лучше всего интерполировать данные (на основе специфичных математических моделей, которые, вероятно, существуют или могут подойти). Например, для каких-то признаков можно сгенерировать псевдослучайные значения, подчиняющиеся некому распределению с учетом других известных признаков. Сюда же можно отнести генерацию синтетических данных (то есть искусственных) на основании собственного понимания предметной области. Долгий и тщательный анализ данных может прояснить поведение скрытого параметра, а также выявить способы, как сэмулировать его с определенной достоверностью. Однако этот подход опасен тем, что неверное понимание данных приведет к тому, что мы заложим в данные другие закономерности и потом их же и найдем при помощи методов машинного обучения. Иными словами, вместо решения исходной задачи мы решим искусственно созданную.

В сложных случаях на практике применяется некоторая комбинация всех вышеперечисленных подходов. Какие-то объекты или столбцы полностью исключаются, потому что их слишком сложно интерполировать или сэмулировать. Как правило, это объекты или признаки с большим числом пропусков (например, где пропусков больше, чем реальных значений). Какие-то признаки помечаются

особым кодом, какие-то данные находятся извне, а какие-то эмулируются синтетическими данными.

Многие нюансы зависят от специфики задачи. Но если никакие идеи не работают, то, вероятно, данных просто недостаточно и нужно получать еще непосредственно исходные данные (проводить физические эксперименты, опрашивать клиентов, измерять какие-то статистические метрики поведения пользователей на веб-сайтах и тому подобное).

Замечание: кроме пропусков данные могут содержать ошибки и выбросы (аномальные отклонения), вызванные неверным заполнением данных или ошибкой измерения. Решение этих проблем выходит за рамки данного учебного пособия и рекомендуется к самостоятельному изучению.

Кодирование нечисловых признаков

Очевидно, что далеко не все признаки объектов естественно описываются численным значением. Если говорить о размерах объекта или стоимости какого-то товара, то эти признаки, несомненно, будут числовыми. Если же речь идет о цвете, типе товара (категории) или вообще о текстовом описании некоторого объекта, то подобные признаки, как правило, поступают неоцифрованными.

Нечисловые признаки с неупорядоченными значениями (в которых между значениями не определена дистанция, то есть нельзя сказать, что больше или меньше) называют категориальными, или номинальными. Типичный подход к их обработке – кодирование категориального признака с m возможными значениями с помощью m бинарных признаков. Каждый бинарный признак соответствует одному из возможных значений категориального признака и является индикатором того, что на данном объекте он принимает данное значение. Такой подход иногда называют one-hot-кодированием. Например, у нас есть три варианта материала изделия: дерево, пластик, сталь. Причем мы не знаем наверняка, что лучше, а что хуже и как это материал влияет на итоговое качество товара. Тогда вместо

того, чтобы закодировать набор матриалов в один столбец как значения 1, 2, 3, one-hot-кодирование даст три столбца и следующие коды: 001, 010, 100. Заметьте, что это не двоичное кодирование.

Если речь идет не просто о категориях, а о целых предложениях или больших текстах на естественном языке, то задача существенно усложняется. Кодирование текста побуквенно в большинстве случаев ничего не даст, так как разнообразие слов и смыслов настолько велико, что на таком уровне абстракции модель не построит нужную функцию. Обработка и понимание текста – это во многом направление для исследований.

Тем не менее на сегодняшний день существует ряд подходов, которые позволяют решать реальные задачи. Первый вариант – отнестись к тексту как к категориям. Допустим, в текстовых данных всего встречается 10 тыс. уникальных слов (не считая союзов, предлогов и тому подобное). Тогда каждое отдельное текстовое описание (присущее конкретному объекту) есть такая категория, где встречаются соответствующие слова, то есть каждый текст будет кодироваться в вектор из 10 тыс. элементов, где на месте соответствующих слов-элементов будут стоять единицы, а на месте слов, которых нет в данном тексте – нули. В итоге получим довольно разреженную матрицу (состоящую в основном из нулей) и при этом довольно большую. Из-за этого возникает проблема ее хранения и обработки. На сегодняшний день существует ряд техник оптимизации работы с разреженными матрицами как на фундаментальном уровне (например, сингулярное разложение), так и на уровне библиотек (например, в пакете scipy).

Однако учитывать все слова часто бывает избыточно и даже бессмысленно. Поэтому на практике применяется подсчет статистических характеристик текста, таких как TF-IDF. Подробнее о статистическом анализе текстов можно прочитать в [3].

Сегодня также актуально кодирование текста методом word2vec на основе машинного обучения. Ключевой особенностью метода является то, что большой массив слов отображается в вещественные

векторы небольшой размерности (100-200 элементов), причем, похожие слова имеют близкие друг к другу векторы (то есть вводится дистанция между словами).

Приведение данных к единому масштабу и стандартизация

«Сырые» данные имеют разный масштаб и разное распределение по каждому признаку. Например, какой-то химический показатель смеси может иметь значения в диапазоне от 0.0001 до 0.2, а другой показатель от –100 до 100. Или, скажем, возраст клиентов может быть от 16 до 40, причем гораздо больше клиентов имеет возраст от 18 до 25, иными словами, математическое ожидание смещено относительно центра распределения. Подобные различия в признаках могут вносить существенную ошибку для множества моделей (например, для регрессии, нейронных сетей), и потому требуется привести все признаки к единому виду.

Существует некоторая путаница в терминах «стандартизация» и «нормализация». Очень часто под стандартизацией и нормализацией понимаются разные вещи, а иногда стандартизацию рассматривают как часть нормализации. Поэтому важно понять общую суть и цель этих методов.

Стандартизация данных – это процесс приведения вектора каждого признака к такому виду, что его математическое ожидание станет нулевым, а дисперсия – единичной.

Нормализация данных – это процесс масштабирования вектора каждого признака, то есть приведение его к такому виду, что вектор будет иметь единичную норму (при этом есть разные способы оценки\подсчета нормы).

Так, стандартизация матрицы X:

[[1., –1.,2.],
[2., 0., 0.],
[0., 1., –1.]],

даст следующий результат:

```
[[ 0.  –1.22  1.34]
[ 1.22   0.   –0.27]
[–1.22  1.22 –1.07]]
```

Видно, что значения вектора сместились (выровнялись относительного единого центра в нуле), а также произошло выравнивание разброса. Уже такого преобразования достаточно, чтобы повысить качество данных. Однако видно, что значения разных векторов не будут в одинаковом диапазоне (от –1 до 1, например). Они будут лишь иметь стандартный разброс в рамках вектора\столбца\признака.

Для того чтобы достичь одинакового масштаба всех векторов, необходима нормализация. Есть несколько видов норм и, соответственно, нормализации.

Самый очевидный и простой метод – это max норма. Чтобы все значения лежали в одном диапазоне, нужно найти максимальное из возможных значений и все остальные поделить на него. Таким образом, максимальное значение будет единицей, а все остальные лягут в диапазон от 0 до 1. Но это при условии, что нет отрицательных значений.

Менее очевидный способ – это L1 норма и нормализация. Формула следующая:

$$x_i' = \frac{x_i}{\|x\|_1} = \frac{x_i}{\sum_j |x_j|},$$

где $\|x\|_1$ и есть L1 норма, а вся формула целиком отображает процесс нормализации вектора *x*.

Еще один способ – это L2 норма. Формула нормализации вектора *x*:

$$x_i' = \frac{x_i}{\|x\|_2} = \frac{x_i}{\left(\sum_j x_j^2\right)^{1/2}},$$

где $\|x\|_2$ и есть L2 норма, а вся формула целиком отображает процесс нормализации вектора *x*.

Исследователи говорят, что лучшие результаты дают L2 и max нормализация, хотя любой вариант лучше, чем использование исходных данных.

По поводу плюсов-минусов этих способов можно сказать следующее: max нормализация не дает запас на неизвестные новые значения (то есть если заранее неизвестен весь диапазон данных, лучше не использовать эту норму), а L2 норма вычислительно дольше, но чаще всего дает оптимальный результат. L1 норма может дать слишком большой запас при большом разбросе данных, что может удалить нужную информацию из данных.

Разметка данных

Разметка данных – последняя из рассматриваемых нами (по порядку, но не по важности) операция с данными. Если речь идет о том, чтобы обучить модель прогнозировать будущие показатели по прошлому опыту (например, прогноз средней выручки по количеству посетителей магазинов), то такие данные, как правило, приходят с известной целевой переменной Y (то есть средней выручкой). Эти данные необходимо будет обработать в соответствии с пунктами выше, но их не потребуется дополнительно размечать. Однако не редкость, когда специалисту по анализу данных необходимо будет самостоятельно размечать выборку. Эта потребность может возникнуть из-за отсутствия размеченных исходных данных, так и может быть обусловлена экспериментами, возникающими в ходе решения общей задачи. Например, для распознавания визуальных образов машин или светофоров потребуется создать специальный файл, в котором будут проставлены ассоциации образов по имени файлов (то есть 001.png – «светофор»). В реальном бизнесе эту задачу может решать и не специалист по анализу данных или машинному обучению, а другие сотрудники (например, со стороны заказчика), но важно отметить, что этот этап может также возникнуть в ходе разработки решения, и его нельзя избежать в случае обучения с учителем.

Однако стоит отметить, что в случаях обучения и без учителя может потребоваться частичная разметка данных с целью тестирования и оценки качества модели, так как в противном случае мы просто получаем «черный ящик» без понимания того, как мы решили поставленную задачу.

Переобучение

Нами были рассмотрены типовые задачи по подготовке данных, теперь рассмотрим одну из основных проблем алгоритмов машинного обучения – переобучение.

Мы помним, что одной из важных характеристик алгоритмов машинного обучения является обобщающая способность. Однако с ней связаны еще два понятия: недообучения и переобучения. При подготовке данного пункта использовался материал (в том числе иллюстративный) из источника [4].

Недообучение возникает при обучении по прецедентам и характеризуется тем, что алгоритм не дает удовлетворительно малой средней ошибки на обучающем множестве. Как правило, это явление появлется вследствие использования недостаточно сложных моделей.

Противоположное этому явлению – переобучение. Его суть состоит в том, что вероятность ошибки натренированного алгоритма на объектах тренировочной выборки оказывается существенно меньше, чем на объектах тестовой. Чаще всего переобучение появляется из-за использования слишком сложных моделей.

Рассмотрим график, иллюстрирующий эффект переобучения (рисунок 5).

Точки на графике с рисунка 5 соответствуют разным способам обучения, и каждая из них получена усреднением по большому числу разбиений исходной выборки объемом в 72 образца на тестовую и обучающую части. Как видно из рисунка, точки имеют постоянное смещение вверх относительно диагонали графика. То есть наблюдается эффект переобучения: ошибки на тестовой выборке появляются чаще, чем на обучающей.

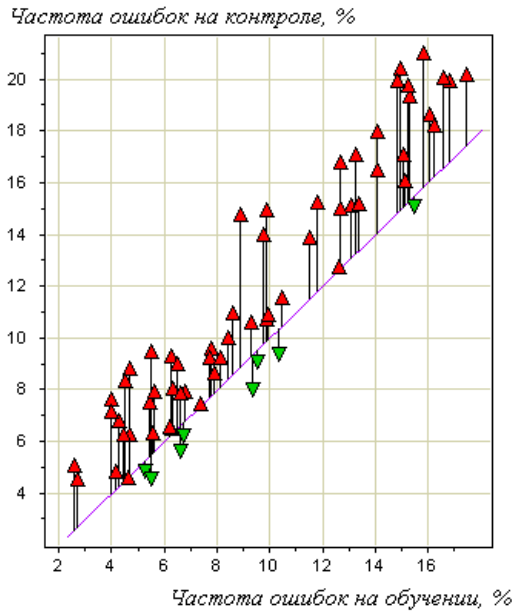


Рисунок 5. Иллюстрация переобучения

Чаще всего при построении алгоритмов обучения используется метод минимизации эмпирического риска (средней ошибки алгоритма на обучающей выборке). Его суть состоит в том, чтобы для текущей модели подобрать алгоритм, минимизирующий значение средней ошибки на данной обучающей выборке.

С переобучением метода минимизации эмпирического риска связаны три утверждения, объясняющие его причину:

1. Минимизация эмпирического риска не является гарантией малой вероятности ошибки на тестовых данных. Можно легко построить алгоритм, который минимизирует эмпирический риск до нуля, однако не будет способен к обучению. Суть состоит в том, что этот алгоритм запоминает обучающую выборку, потом сравнивает запомненный образец с предъявляемым. При совпадении предъявленного объекта и образца обучающей выборки алгоритм выдаст правильный ответ, иначе – выведется произвольный. То есть эмпирический риск равен нулю, однако обобщающей способности у алгоритма нет.

2. Согласно [4]: «Переобучение появляется именно вследствие минимизации эмпирического риска. Пусть задано конечное множество из D алгоритмов, которые допускают ошибки независимо и с одинаковой вероятностью. Число ошибок любого из этих

алгоритмов на заданной обучающей выборке подчиняется одному и тому же биномиальному распределению. Минимум эмпирического риска – это случайная величина, равная минимуму из D независимых одинаково распределенных биномиальных случайных величин, ожидаемое значение которой уменьшается с ростом D. Соответственно, с ростом D увеличивается переобученность – разность вероятности ошибки и частоты ошибок на обучении.

В данном модельном примере легко построить доверительный интервал переобученности, так как функция распределения минимума известна. Однако в реальной ситуации алгоритмы имеют различные вероятности ошибок, не являются независимыми, а множество алгоритмов, из которого выбирается лучший, может быть бесконечным. По этим причинам вывод количественных оценок переобученности является сложной задачей, которой занимается теория вычислительного обучения. До сих пор остается открытой проблема сильной завышенности верхних оценок вероятности переобучения».

3. Переобучение появляется в связи с избыточной сложностью модели. Всегда можно найти оптимальное значение сложности модели, при котором переобучение будет минимальным. Для примера приведем несколько графиков, на которых будет видна зависимость переобучения от сложности модели. При степени 2 полинома (рисунок 6) модель является недообученной. При степени 40 (рисунок 7) – переобученной и неустойчивой, а вот степень 20 (рисунок 8) – оптимальна.

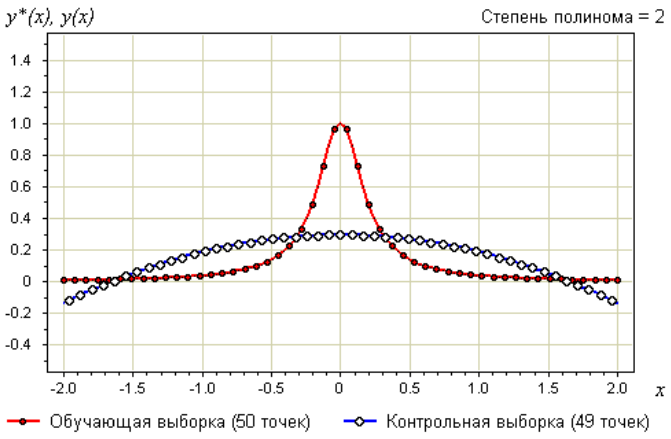


Рисунок 6. Недообученная модель

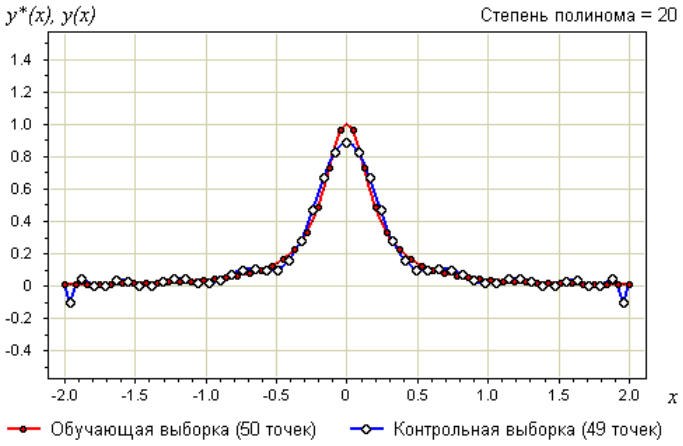


Рисунок 7. Оптимальная модель

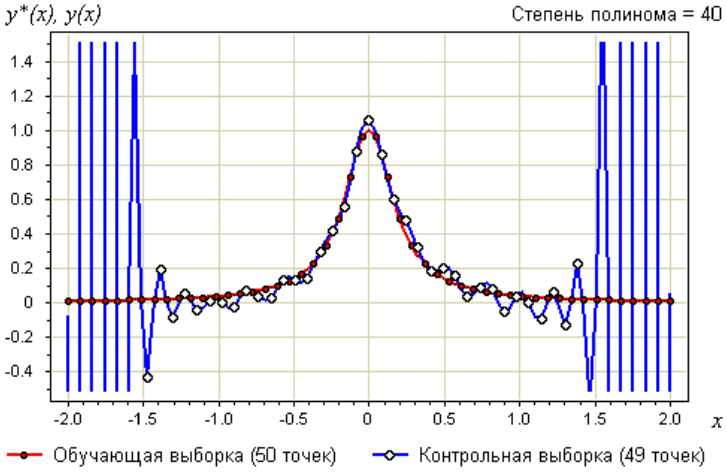


Рисунок 8. Переобученная модель

Одним из способов измерения вероятности переобучения является эмпирический метод Монте-Карло, или метод скользящего контроля. В литературе также встречаются названия кросс-проверка, или кросс-валидация. Суть его в следующем: производится некоторое количество разбиений исходной выборки на обучающую и контрольную. Для каждого разбиения происходит обучение алгоритма на обучающей подвыборке и оценка средней ошибки на контрольной. Затем вычисляется оценка скользящего среднего, как средняя по всем разбиениям величина вычисленной ошибки. Для независимой выборки этот показатель дает несмещенную оценку вероятности ошибки. Метод скользящего контроля является стандартным способом сравнения и оценивания алгоритмов классификации, регрессии, прогнозирования.

При использовании кросс-валидации можно сделать следующие выводы:

1. Если ошибка большая на большинстве участков, то скорее всего проблема в модели.

2. Если данные обучающей выборки характеризуются сильно смещенными математическим ожиданием и дисперсией, то уровень обобщения будет низким, что может быть связано с переобучением.

3. Если найдены сильные отклонения на определенных подвыборках, то, вероятно, проблема в этих участках данных или модель недообучается.

4. При малом объеме обучающей выборки кросс-валидация может стать способом борьбы с переобучением.

Если же говорить о прямых способах борьбы с переобучением, то можно выделить следующие:

1. Упрощение модели.

2. Подготовка большего числа обучающих данных (возможно, с помощью генерации).

3. Регуляризация.

Остановимся подробнее на последнем. Регуляризация представляет собой добавление некоторой дополнительной информации к условию минимизации ошибки. Выполняется это, чтобы решить некорректно поставленную задачу или предотвратить переобучение. Чаще всего добавляемая информация принимает вид штрафа за сложность модели. Например, введенные ограничения по норме векторного пространства или гладкости результирующей функции. Или же, с байесовской точки зрения, добавленные априорные распределения на параметры модели.

Согласно [5]: «Существуют следующие основные виды регуляризации:

L1-регуляризация (англ. Lasso regression):

$$\mathrm{RSS}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i,\mathbf{w}) - y_i)^2 + \lambda\sum_{j=0}^{M}|w_j| \ ,$$

где w – вектор весов полинома;

λ – коэффициент регуляризации.

L2, или Регуляризация Тихонова (в английской литературе – ridge regression или Tikhonov regularization), для интегральных уравнений позволяет балансировать между соответствием данным и маленькой нормой решения:

$$\mathrm{RSS}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}(f(x_i,\mathbf{w}) - y_i)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2,$$

где $\|W\|^2$ – квадратичная норма вектора весов (сумма квадратов каждого веса).

Иными словами, переобучение в большинстве случаев проявляется в том, что в получающихся многочленах слишком большие коэффициенты. Соответственно и бороться с этим можно довольно естественным способом: нужно просто добавить в целевую функцию штраф, который бы наказывал модель за слишком большие коэффициенты».

По поводу применения той или иной регуляризации приведем материал источника [6]: «Регуляризацию можно применять с любым методом МО-классификации, который основан на математическом уравнении. Примеры включают линейную, логистическую регрессию и нейронные сети. Поскольку это уменьшает величину весовых значений в модели, регуляризацию иногда называют сокращением весов. Основное преимущество применения регуляризации в том, что оно часто приводит к созданию более точной модели. Главный недостаток заключается во введении дополнительного параметра, значение которого нужно определить, – весового значения регуляризации. В случае логистической регрессии это не слишком серьезно, так как в этом алгоритм обычно используется лишь параметр скорости обучения, но при использовании более сложного метода классификации, в частности нейронных сетей, добавление еще одного так называемого гиперпараметра может потребовать массы дополнительной работы для подбора комбинации значений двух параметров.

L1- и L2-регуляризация – процессы схожие. Какой же из них лучше? В принципе, исследователями сформулированы кое-какие теоретические правила насчет того, какая регуляризация лучше для определенных задач, но на практике придется поэкспериментировать, чтобы найти, какой тип регуляризации лучше в вашем случае и стоит ли вообще использовать какую-либо регуляризацию.

Применение L1-регуляризации иногда может давать полезный побочный эффект, вызывающий стремление одного или более весовых значений к 0.0, а это означает, что соответствующий признак не оказывает значимого влияния на результирующий, то есть включение его в модель требуется. Это одна из форм того, что называют «селекцией признаков». В отличие от L1, L2-регуляризация ограничивает весовые значения модели, но обычно не приводит к полному обнулению этих значений. Поэтому может показаться, что L1-регуляризация лучше L2-регуляризации. Однако недостаток применения L1-регуляризации в том, что этот метод не так-то просто использовать с некоторыми алгоритмами машинного обучения. Например, с теми, в которых используются численные методы для вычисления так называемого градиента. L2-регуляризацию можно использовать с любым типом алгоритма обучения.

Таким образом, можно сделать вывод, что L1-регуляризация иногда дает полезный побочный эффект удаления ненужных признаков, присваивая связанным с ними весам значение 0.0, но L1-регуляризация стабильно работает не со всеми формами обучения. L2-регуляризация работает со всеми формами обучения, но не обеспечивает неявной селекции функций. На практике же следует использовать метод проб и ошибок, чтобы определить, какая форма регуляризации (если она вообще нужна) лучше для конкретной задачи».

Рассмотренные методы борьбы с переобучением подойдут для регрессии или нейронных сетей. Для борьбы же с переобучением в моделях Деревьев Решений используют pruning (так называемое усечение) – удаление наименее информативных узлов для упрощения модели. Дело в том, что сразу нельзя построить оптимальное

(маленькое) дерево, так как в дереве малого размера будет мало информативных параметров, чтобы корректно обработать все входные образы (модель будет неполной, неинформативной). Поэтому сперва делают большое, но информативное дерево, а затем удаляют наименее информативные узлы и подтягивают дерево, уменьшая его размер\глубину.

Модели и алгоритмы машинного обучения

Как было сказано во введении, машинное обучение находится на стыке математической статистики, численных методов оптимизации, теории вероятностей и дискретного анализа. Эта дисциплина комбинирует и использует различные методы в построении математических моделей объектов и явлений для последующего их изучения и добычи знаний. В данном разделе предлагается рассмотреть, основные элементы математики, использующиеся в машинном обучении, а также изучить методы анализа данных, которые заложили основу рассматриваемой дисциплины.

Первое, что необходимо отметить, – любая дисциплина, использующая математический аппарат, так или иначе занимается математическим моделированием – заменой реального объекта его абстрактным, идеализированным представлением и использованием полученного представления для изучения объекта и добычи знаний. То есть основное, с чем работает машинное обучение, – это математические модели. Их можно разделить на различные типы по некоторым признакам. С точки зрения математического вида функции, составляющей модель, выделяют линейные и нелинейные модели. По количеству переменных в модели они делятся на сосредоточенные и распределенные. С точки зрения учета случайности модели делятся на детерминированные и стохастические, а с точки зрения изменчивости во времени – на статические и динамические. По природе используемых в модели параметров и переменных их можно разделить на дискретные и непрерывные.

В общем виде математическая модель может быть представлена следующим образом:

$$Y=F(X,W_c,W_d,W_s),$$

где Y – выходные данные; X – входные данные; F – некоторая функция; W_c – константные параметры модели; W_d – динамические параметры модели; W_s – статические параметры модели.

При построении моделей элементы множеств *W*, как правило, заранее неизвестны и требуют нахождения. Есть два способа решения этой задачи: аналитический и численный. Первый предполагает нахождение корней, второй – приближение результата к некоторому удовлетворительному значению. Удовлетворительность в данном случае оценивается некоторым заранее определенным критерием. При этом для решения задачи численным методом нередко происходит ее переформулировка для определения и введения условия, определяющего качество решения. Необходимо отметить, что численные методы чаще всего опираются на отклонение получаемого результата от желаемого и стремятся минимизировать это отклонение. Причем данный подход будет работать, даже если неизвестна точная формула для получения выходного значения, однако обязательно существует способ измерения отклонения, а также можно найти значения, которые он должен принимать.

Появление численных методов оптимизации было обусловлено, во-первых, тем, что не все задачи можно решить аналитически. Классический пример подобной задачи – гравитационная задача *N* тел. Согласно Википедии [1], ее формулировка звучит так: «В пустоте находится *N* материальных точек, массы которых известны $\{m_i\}$. Пусть попарное взаимодействие точек подчинено закону тяготения Ньютона, и пусть силы гравитации аддитивны. Пусть известны начальные на момент времени $t=0$ положения и скорости каждой точки $\mathbf{r}_i|_{t=0} = \mathbf{r}_{i0}$, $\mathbf{v}_i|_{t=0} = \mathbf{v}_{i0}$. Требуется найти положения точек для всех последующих моментов времени». И согласно тому же источнику [1]: «На данный момент в общем виде задача *N* тел для $N>3$ может быть решена только численно, причем для $N=3$ ряды Зундмана даже при современном уровне компьютеров использовать практически невозможно».

Во-вторых, даже возможные аналитические решения систем могут обладать существенной сложностью. Матричные решения имеют вычислительную сложность *O(N^3)*, тогда как численные – линейную.

В-третьих, при противоречивости или мультиколлинеарности данных аналитические решения могут давать неопределенный результат. В отличие от них численные методы всегда сходятся пусть и к локальному, но все же определенному решению.

В основном, при решении задач минимизации ошибки, в машинном обучении используются градиентные методы, составляющие особый класс алгоритмов оптимизации. Градиент в данном случае – это (согласно Википедии [1]): «вектор, своим направлением указывающий направление наибольшего возрастания некоторой величины φ, значение которой меняется от одной точки пространства к другой (скалярного поля), а по величине (модулю) равный скорости роста этой величины в этом направлении.

Например, если взять в качестве φ высоту поверхности земли над уровнем моря, то ее градиент в каждой точке поверхности будет показывать «направление самого крутого подъема», и своей величиной характеризовать крутизну склона.

С математической точки зрения на градиент можно смотреть как на:

1. Коэффициент линейности изменения значения функции многих переменных от изменения значения аргумента.

2. Вектор в пространстве области определения скалярной функции многих переменных, составленный из частных производных.

3. Содержимое Матрицы Якоби. Ее строки содержат градиенты составных скалярных функций, из которых состоит векторная функция многих переменных.

Пространство, на котором определена функция и ее градиент, может быть как обычным трехмерным пространством, так и пространством любой другой размерности, любой физической природы или чисто абстрактным (безразмерным)».

Метод градиентного спуска – нахождение локального экстремума (минимума или максимума) функции путем движения вдоль градиента в направлении наискорейшего спуска, задаваемого антиградиентом. Существуют следующие типы градиентного спуска:

1. С постоянным шагом.
2. С дроблением шага.
3. Наискорейшего спуска.
4. Стохастический.

Более подробно о градиентном спуске мы поговорим далее, здесь же рассмотрим аналитические методы, используемые в машинном обучении, так как именно они составляют первое поколение алгоритмов, применяемых для анализа данных.

Методы теории вероятностей

Первый срез методов, который необходимо рассмотреть, – методы теории вероятностей. Теория вероятностей – раздел математики, в котором изучаются случайные величины и события, их свойства и возможные операции над ними. Таким образом, ключевое понятие, лежащее в основе этой дисциплины, – вероятность события P_i. Опираясь на него, можно дать определение полной группы событий. Она определяется как система случайных событий, которая обладает следующими свойствами:

1. В результате случайного эксперимента непременно произойдет одно и только одно из составляющих ее событий.
2. Сумма вероятностей всех событий полной группы равна 1.

Достаточно часто при исследовании данных (выборок) и подсчете определенных характеристик выборки считаются полными группами событий.

С помощью методов теории вероятностей можно проводить как простые, так и сложные операции по анализу данных. Среди простых можно выделить подсчеты вероятностных характеристик выборки: медианы, математического ожидания, дисперсии, а также величины среднеквадратического отклонения. Рассмотрим их подробнее.

Медиана – число, характеризующее выборку по среднему из ее значений. То есть, если все данные выборки различны, и она упорядочена по возрастанию, то ровно половина из элементов выборки

будет меньше медианы, и ровно половина – больше. Формально величину можно выразить следующим образом:

Если $X=\{x_{1\ldots}x_n\}$ – характеризуемая выборка, то x_k – медиана, если $x_j<x_k$, при $j \in [1,k)$ и $x_k<x_i$, при $i \in (k,n]$ и $k=n/2$.

Рассмотрим пример. Пусть выборка X = {1, 2, 3, 4, 5, 6, 7, 8, 9}, тогда ее медианой будет число 5.

Математическое ожидание – среднее значение вероятностных элементов выборки. Формально она рассчитывается так:

$$M|X|=\sum x_i p_i \ .$$

Рассмотрим пример. Пусть выборка X = {1, 2, 3, 4, 5, 6, 7, 8, 9}, тогда ее математическим ожиданием будет величина, равная 5. Вычисляется она следующим образом: выборка рассматривается как полная группа событий с равными вероятностями. То есть p_i для каждого элемента равно 0,11. Тогда 1×0,11+2×0,11+3×0,11…=5.

Дисперсия – мера разброса элементов выборки относительно ее математического ожидания. Рассчитывается данная величина по формуле

$$D|X|=\sum(x_i - M|X|)^2 p_i \ .$$

При этом p_i рассчитывается как $1/(n-1)$, где n – количество элементов выборки. Это выполняется для того, чтобы не учитывать отклонение самого математического ожидания от себя. То есть, для выборки X = {1, 2, 3, 4, 5, 6, 7, 8, 9} дисперсия будет рассчитываться следующим образом:

(1–5) ×(1–5) ×0,125+(2–5) ×(2–5) ×0,125+(3–5) ×(3–5) ×0,125…=7,5.

Среднеквадратическое отклонение – величина, характеризующая рассеивание значений выборки относительно ее математического ожидания. Формула расчета

$$\sigma = \sqrt{\frac{1}{n}\sum(x_i - \bar{x})^2},$$

где n – количество элементов в выборке; $\bar{x}$ – математическое ожидание.

Интерпретировать данную величину можно следующим образом: чем больше значение среднеквадратического отклонения, тем

больший разброс значений в представленном множестве (относительно его средней величины). Малое значение среднеквадратического отклонения говорит о том, что элементы выборки сгруппированы вокруг ее среднего значения. Если говорить о более практическом применении величины, то в экономике, например, она характеризует доходность портфеля и его риск.

Однако рассмотренные показатели характеризуют случайную величину лишь с какой-то одной стороны. Наиболее же полно и исчерпывающе ее описывает закон распределения – функция, определяющая для выборки X вероятность попадания в некоторый интервал или вероятность получения определенного значения x_i. Если какой-либо закон распределения описывает случайную величину, то говорят, что она ему подчиняется или по нему распределена. Иными словами, закон распределения описывает область значений случайной величины и вероятности их получения.

Среди законов распределения наиболее часто используется закон нормального распределения или распределения Гаусса, который характеризует большинство процессов, встречающихся в мире. Отсюда и произошло его название (нормальное). Закон (плотность распределения вероятности) описывается следующей формулой:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}},$$

где σ – среднеквадратическое отклонение; μ – математическое ожидание.

С математической точки зрения можно заметить, что данная функция зависит от двух параметров: математического ожидания и среднеквадратического отклонения, поэтому по сути данный закон представляет собой семейство распределений. В качестве примера приведем изображение из источника [7], показывающее графики нормальных распределений с разными параметрами (рисунок 9).

Обратите внимание!

- *График с параметрами σ2=1 и μ=0 соотвествует стандартному нормальному распределению.*

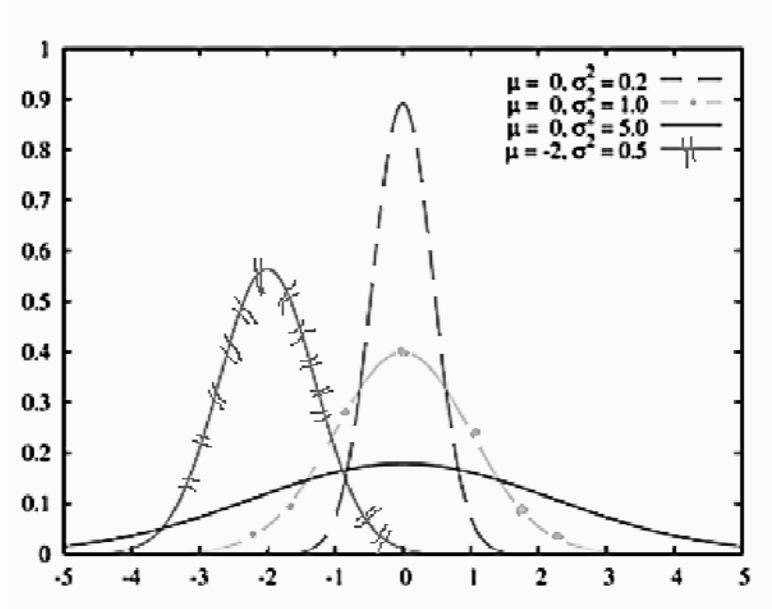


Рисунок 9. Плотность нормального распределения

Согласно Википедии [1]: «Важное значение нормального распределения во многих областях науки (например, в математической статистике и статистической физике) вытекает из центральной предельной теоремы теории вероятностей. Если результат наблюдения является суммой многих случайных слабо взаимозависимых величин, каждая из которых вносит малый вклад относительно общей суммы, то при увеличении числа слагаемых распределение центрированного и нормированного результата стремится к нормальному. Этот закон теории вероятностей имеет следствием широкое распространение нормального распределения, что и стало одной из причин его наименования».

Кроме описанных выше величин и характеристик для анализа данных очень часто используется одна из основных теорем теории вероятностей – теорема Байеса.

Она позволяет определить вероятность наступления какого-либо события при условии, что произошло другое событие, статистически

взаимосвязанное с исследуемым. Ключевыми понятиями в данной теореме являются понятия априорной и апостериорной вероятностей. Рассмотрим их подробнее.

Априорная вероятность – назначенная событию вероятность, при условии отсутствия знаний, поддерживающих его наступление. Апостериорная вероятность – назначенная событию вероятность при условии наличия знаний, поддерживающих его наступление и полученных опытным путем.

Теперь перейдем непосредственно к теореме Байеса. Формулируется она следующим образом:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)},$$

где $P(A|B)$ – апостериорная вероятность справедливости гипотезы A при наступлении B; $P(B|A)$ – вероятность наступления события B при истинности гипотезы A; $P(A)$ – априорная вероятность гипотезы A; $P(B)$ – вероятность наступления события B.

Метод Байеса имеет свои достоинства и недостатки. К первому можно отнести возможность использования экспертных знаний, возможность точного описания явления и хорошую интерпретируемость результатов. К недостаткам же относится следующее:

1. Метод не ставит целью минимизацию ошибки классификации.
2. Метод требует работы эксперта.
3. Сильная зависимость результатов от выбора модели.
4. Плохая работа при малом количестве и высокой размерности данных.
5. Метод дает плохое обобщение, особенно на высокоуровневых признаках.
6. Метод дает плохие результаты при взаимозависимости признаков.

Таким образом, можно сделать вывод, что данный метод хорошо работает в тех случаях, где признаки и результат сильно завязаны на их частотных характеристиках.

Деревья решений

Следующий метод анализа данных, который нам необходимо рассмотреть, – применение деревьев решений. Дерево решений – средство поддержки принятия решений для прогнозных моделей. Суть его работы заключается в последовательном разбиении множества данных на непересекающиеся классы, которые в свою очередь также подвергаются разбиению по каким-либо критериям с оценкой эффективности разбиения. Как правило, дерево решений состоит из «узлов», «листьев» и «веток». «Ветки» содержат записи атрибутов, от которых зависит целевая функция, «листья» – значения целевой функции, а «узлы» – остальные атрибуты, по которым происходит классификация. Чаще всего выделяют два типа деревьев: для классификации (в этом случае предсказываемый результат – класс, которому принадлежат данные) и для регрессии (результат – прогнозируемое значение целевой функции).

Обобщенный алгоритм построения дерева решений по обучающей выборке состоит из следующих шагов:

1. Берем следующий атрибут и помещаем его в корень.
2. Для всех значений этого атрибута – оставляем в «листьях» данной «ветки» только те значения, которые соответствуют определенному условию.
3. Продолжаем строить дерево среди оставленных на предыдущем шаге «листьев».

Для выбора следующего атрибута может быть использован один из следующих основных алгоритмов:

1. ID3. Атрибут выбирается на основе прироста информации и минимизации энтропии. Напомним, что под энтропией в данном случае подразумевается мера неупорядоченности системы. Чем меньше ее величина, тем более упорядочены составляющие системы.
2. C4.5 – улучшенная версия предыдущего алгоритма, в которой используется нормализованный прирост информации.

3. CART – алгоритм, используемый для построения бинарных деревьев, производящий разбиение на основе модальных значений признаков.

Рассмотрим работу по построению дерева решений с использованием первого алгоритма. Более подробно об этом можно прочитать в [8, 9].

Кратко алгоритм ID3 может быть описан тремя шагами:

1. Посчитать энтропию разбиваемого множества.
2. Выбрать признак с минимальной энтропией и, соответственно, максимальной информационной выгодой.
3. На основе полученного признака создать узел дерева и повторить процедуру.

Рассмотрим построения дерева решений для классификации следующего множества из семи объектов: {красный круг, зеленый квадрат, красный квадрат, зеленый треугольник, зеленый круг, красный треугольник, красный прямоугольник}. Как мы видим, каждый из объектов характеризуется двумя признаками: цвет и форма, то есть именно по ним мы будем проводить разбиение. Для меры энтропии выберем формулу энтропии Шеннона:

$$H = -\sum_{i=1}^{N} p_i \times \log p_i$$

В таблице 2 представлены расчеты энтропии всей системы. Подсчеты энтропии для каждого из признаков приведены в таблицах 3 и 4, соответственно.

Таблица 2. Подсчет энтропии системы

Объект	**pi**	**log(pi)**	**pi*log(pi)**
красный квадрат	0,142857	-0,8451	-0,12073
красный прямоугольник	0,142857	-0,8451	-0,12073
красный круг	0,142857	-0,8451	-0,12073
зеленый квадрат	0,142857	-0,8451	-0,12073
зеленый треугольник	0,142857	-0,8451	-0,12073
зеленый круг	0,142857	-0,8451	-0,12073
красный треугольник	0,142857	-0,8451	-0,12073
Энтропия			0,845098

Таблица 3. Подсчет энтропии для признака «Цвет»

Объект	pi	log(pi)	pi*log(pi)
красный	0,571429	-0,24304	-0,13888
зеленый	0,428571	-0,36798	-0,1577
Энтропия			0,296583

Таблица 4. Подсчет энтропии для признака «Форма»

Объект	pi	log(pi)	pi*log(pi)
круг	0,285714	-0,54407	-0,15545
квадрат	0,285714	-0,54407	-0,15545
треугольник	0,285714	-0,54407	-0,15545
прямоугольник	0,142857	-0,8451	-0,12073
Энтропия			0,587072

Как мы видим, первым признаком, на основе которого будет происходить разбиение, станет цвет, а вторым – форма. Классификация исходного множества с помощью построенного дерева показана на рисунке 10.

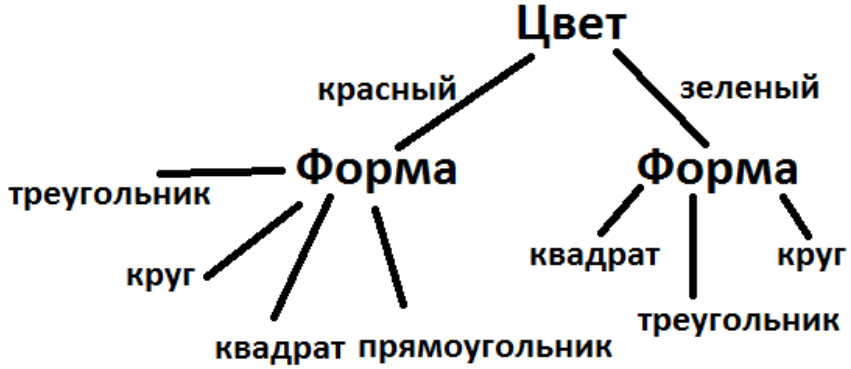


Рисунок 10. Классификация фигур с помощью дерева

Стоит отметить, что чаще всего деревья решений – бинарные. В узлах они содержат условия, а ветви соответствуют истинности или ложности этого условия. Поэтому, если дерево с рисунка 10 преобразовать в классическое дерево решений, то получится изображение, представленное на рисунке 11.

Теперь рассмотрим использование этого метода для решения задачи из источника [10]: спрогнозируем исход матча для футбольной команды исходя из следующих параметров:

- выше ли находится соперник по турнирной таблице;
- дома ли играется матч;
- пропускает ли матч кто-либо из лидеров команды;
- идет ли дождь.

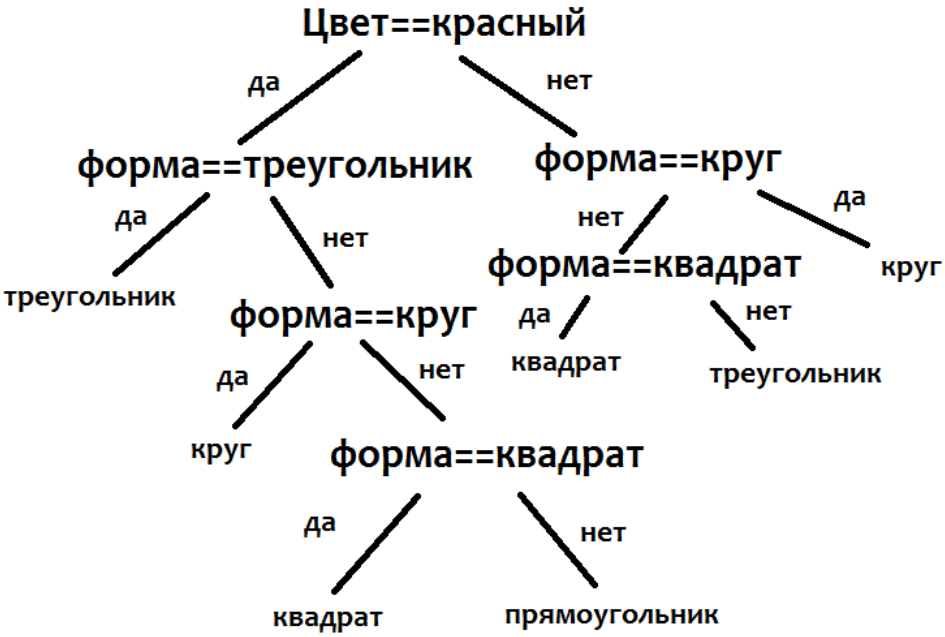


Рисунок 11. Классическое дерево решений

Прогноз предлагается выполнить на основе статистики, представленной в таблице 5.

Таблица 5. Статистические данные для задачи

Соперник	Играем	Лидеры	Дождь	Победа
Выше	Дома	На месте	Да	Нет
Выше	Дома	На месте	Нет	Да
Выше	Дома	Пропускают	Нет	Нет
Ниже	Дома	Пропускают	Нет	Да
Ниже	В гостях	Пропускают	Нет	Нет
Ниже	Дома	Пропускают	Да	Да
Выше	В гостях	На месте	Да	Нет
Ниже	В гостях	На месте	Нет	???

Первое, что мы рассчитываем, – энтропию для признаков. Результаты можно увидеть в таблице 6, причем данные приведены в округленном виде.

Таблица 6. Энтропия для признаков

Соперник	Играем	Лидеры	Дождь	Победа
0,29	0,26	0,29	0,29	0,29

Согласно таблице 6, первый признак, по которому будет происходить деление, – место игры. По нашим данным получатся два множества: со значением «дома» и со значением «в гостях». Если мы рассмотрим второе множество, то увидим, что следующий целевой признак с нулевой энтропией – исход матча: все элементы множества «в гостях» имеют значение поля победа «нет». Следовательно, по нашим статистическим данным с помощью деревьев решений мы прогнозируем поражение.

Если говорить о достоинствах деревьев решений, то можно выделить следующие. Во-первых, простота понимания и интерпретации. Во-вторых, минимальные требования к подготовке данных, а также способность работы с большими объемами данных. В-третьих, метод одинаково хорошо работает с разными видами признаков. В-четвертых, является надежным методом и позволяет оценить модель статистическими тестами. К недостаткам же можно отнести следующее:

1. Достаточно сложно построить оптимальное дерево решений.
2. Подверженность переобучению.
3. Не для всех задач может быть получено решение удовлетворительного качества.

Статистические модели и методы

Мы рассмотрели методы машинного обучения, предоставляемые теорией вероятностей, а также модель дерева решений. Теперь обратимся к статистике и кратко опишем ее модели и методы, которые могут быть использованы для решения задач машинного обучения, а именно методы регрессионного анализа, подробно рассмотренные в следующих источниках: [11,12,13].

В контексте машинного обучения под регрессионным анализом понимается процесс построения математической модели, описывающей зависимость некоторой целевой характеристики объекта или процесса от других его характеристик. Например, зависимость числа новых клиентов от величины зарплаты работающего на улице промоутера.

В задаче регрессионного анализа всегда есть обучающая выборка, состоящая из входных параметров и откликов, а также начальная параметрическая модель, в самом простом случае – линейная, однако не обязательно таковая. Для задачи из примера эта модель может иметь вид $y=\beta_0+\beta_1\times x$, где x – размер зарплаты промоутера; y – количество новых клиентов; β_0 и β_1 – параметры модели. Задача регрессии – оценить их, то есть найти такие значения β_0 и β_1, чтобы полученная модель отражала зависимость между входом и выходом с требуемой точностью.

После получения адекватной модели мы можем решать задачу прогнозирования, подставляя в полученную формулу величину x и вычисляя величину y (с удовлетворяющей нас погрешностью). Приведенный выше пример модели – модель парной линейной регрессии, но помимо нее существует и множественная, и нелинейная регрессии. Начнем рассмотрение с самого простого.

Допустим, мы хотим описать зависимость между двумя факторами моделью вида $y=\beta_0+\beta_1\times x$. Первое, что необходимо учесть, – построенная линия никогда не будет точно проходить по опытным точкам, поэтому истинный вид регрессионной модели будет $y=\beta_0+\beta_1\times x+e$, где e – ошибки наблюдений. Второе – прежде чем переходить к оцениванию параметров модели, целесообразно построить диаграмму рассеяния, чтобы убедиться, что выбранная модель действительно может описать зависимость между факторами. На диаграмме рассеяния каждой паре «зависимый-влияющий параметр» соответствует точка на плоскости. Как правило, зависимый фактор откладывается по оси ординат, а второй – по оси абсцисс. Модель парной линейной регрессии графически представляет собой линию, следова-

тельно, использовать ее для описания зависимости целесообразно, если на диаграмме рассеяния точки располагаются вокруг (и достаточно близко) какой-либо прямой. Отклонение реальных точек от модельных – остатки или ошибки наблюдений, которые обозначаются *e*. Пример диаграммы рассеяния показан на рисунке 12.

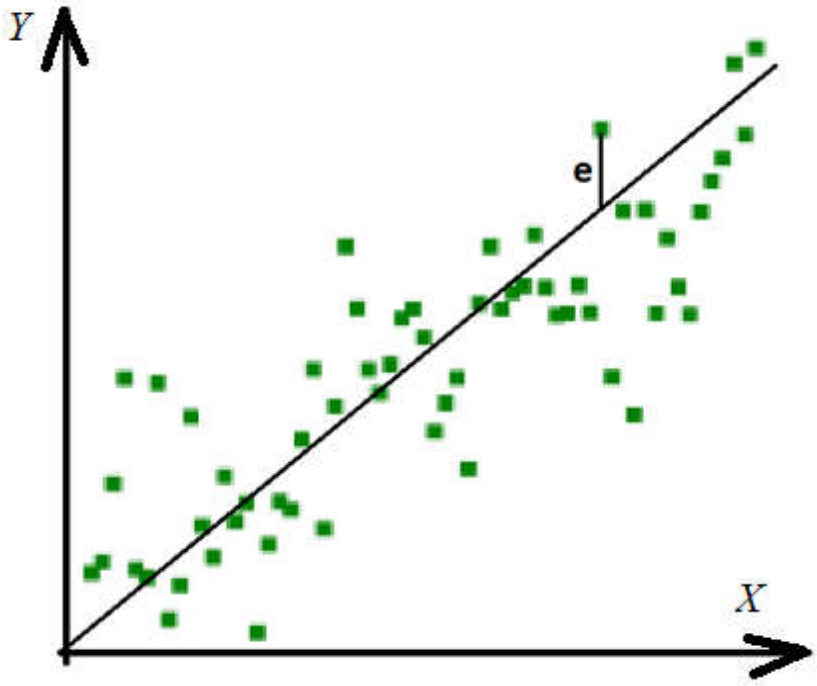


Рисунок 12. Диаграмма рассеяния

Если после анализа диаграммы рассеяния принимается решение использовать линейную парную регрессию для моделирования зависимости, то следующим шагом будет нахождение параметров модели β_0 и β_1. Для решения этой задачи в девяносто девяти процентах случаев используется метод наименьших квадратов, предложенный Гауссом более двухсот лет назад. Суть данного метода заключается в минимизации суммы квадратов отклонений опытных данных от модельных. Формально эта задача описывается так:

$$Q = \sum e_i^2 \rightarrow min$$

В данном случае e_i вычисляется следующим образом:

$$e_i = \beta_0 + \beta_1 * x_i - y_i,$$

где y_i – реальное выходное значение для входного значения x_i.

Чтобы решить указанную задачу оптимизации, решают систему уравнений:

$$\left.\begin{aligned}\frac{\partial Q}{\partial \beta_0} &= 0\\ \frac{\partial Q}{\partial \beta_1} &= 0\end{aligned}\right\}$$

Найдя необходимые частные производные и выполнив преобразования по упрощению выражений, получают нормальную систему уравнений для парной линейной регрессии:

$$\left.\begin{aligned} n * \beta_0 + \beta_1 * \sum x_i - \sum y_i &= 0\\ \beta_0 * \sum x_i + \beta_1 * \sum x_i^2 - \sum x_i y_i &= 0\end{aligned}\right\}$$

Рассмотрим для примера построение парной линейной регрессии для задачи зависимости ежемесячного количества новых клиентов от почасовой зарплаты промоутера по данным, представленным в таблице 7.

Таблица 7. Данные для анализа

Зарплата (*x*)	100	120	130	150
Количество клиентов (*y*)	70	100	120	140

Подставим данные в систему уравнений и получим следующий ее вид:

$$\left.\begin{aligned} 4 \times \beta_0 + \beta_1 \times 500 - 430 &= 0\\ \beta_0 \times 500 + \beta_1 \times 63800 - 55600 &= 0\end{aligned}\right\}$$

Выразим β_0 из первого уравнения:

$$\beta_0 = 107,5 - \beta_1 \times 125$$

Подставим во второе и найдем β_1:

$$53750 - 62500 \times \beta_1 + \beta_1 \times 63800 - 55600 = 0$$

$$\beta_1 = \frac{1850}{1300}.$$

В итоге получим уравнение регрессии:

$$y = 1,423\,1x - 70,385\ .$$

Полученные коэффициенты можно интерпретировать следующим образом. β_0 – значение при x=0. То есть, если у нас не будет работать промоутер (то есть мы не будем платить ему зарплату), ежемесячно мы прогнозируем отток 70 клиентов. Содержательная интерпретация второго коэффициента следующая: каждый рубль в зарплате промоутера дает 1,4 нового клиента в месяц.

Необходимо отметить, что коэффициенты для парной линейной регрессии можно найти средствами табличного процессора Excel. Для этого строится обычная точечная диаграмма, а затем отображается линия тренда и уравнение, как показано на рисунке 13.

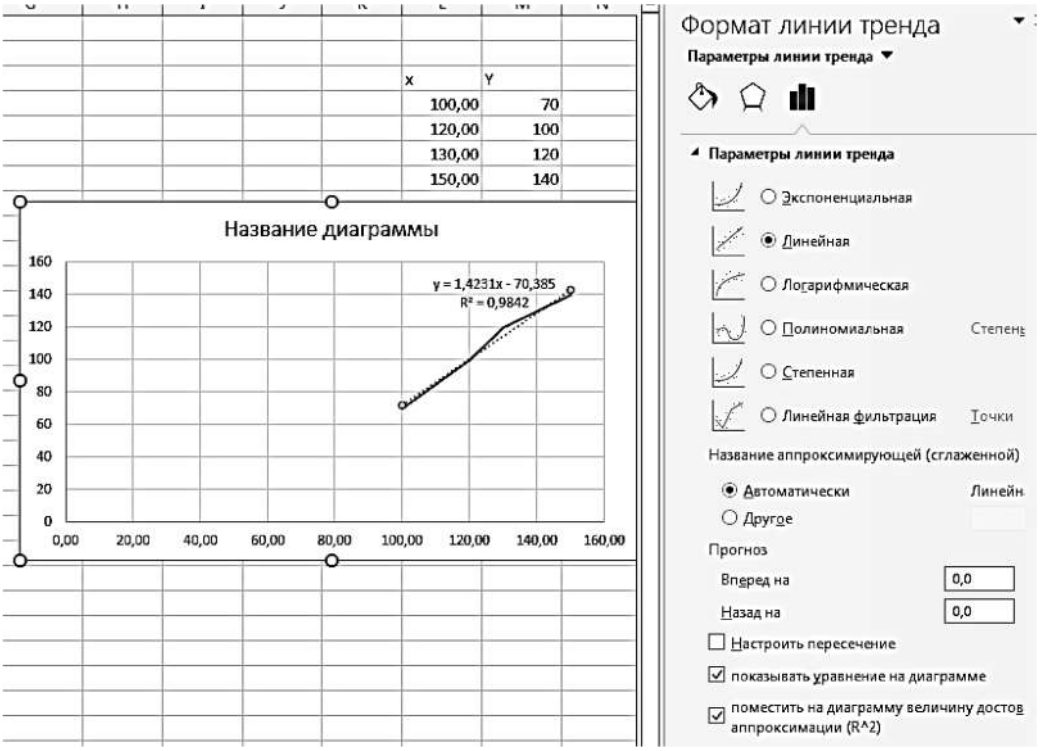


Рисунок 13. Работа с парной линейной регрессией в Excel

Там же можно отобразить величину достоверности аппроксимации (или коэффициент детерминации R^2), которая показывает, насколько модельные значения близки к реальным. Чем ближе эта величина к 1, тем лучше модель описывает реальность. То есть, по сути, это критерий качества модели. Рассчитывается он по формуле

$$R^2 = \frac{\sum \tilde{y}_i^2 - n\bar{y}^2}{\sum y_i^2 - n\bar{y}^2},$$

где $\bar{y}=\frac{\sum y_i}{n}$ – среднее значение реальных результатов, $\tilde{y}$ – прогнозное значение.

Областью значения данной величины будет отрезок от 0 до 1.

Кроме расчета величины достоверности аппроксимации есть еще несколько способов оценить качество регрессионной модели. Однако в данном пособии не будем подробно останавливаться на этих методах, а лишь перечислим их:

1. Анализ диаграммы рассеяния;
2. Проверка статистических гипотез о значимости модели;
3. Анализ остатков.

По третьему пункту поясним: метод наименьших квадратов справедлив, если остатки обладают следующими свойствами:

1. Нормальным распределением.
2. Взаимонезависимостью.
3. Нулевым математическим ожиданием.
4. Постоянной дисперсией.

То есть, если остатки разбросаны хаотично, то метод наименьших квадратов можно использовать для решения поставленной задачи.

Если не хаотично, то нужно смотреть: если остатки растут при росте *x*, то есть их график напоминает диаграмму рассеяния, показанную на рисунке 12, это говорит о гетероскедастичности остатков. В данном случае необходимо использовать взвешенный метод наименьших квадратов. Если график остатков похож на отрезок параболы, то скорее всего была выбрана неправильная модель и необходимо выбрать другую. Если график остатков напоминает, например, синусоиду, то есть они циклически колеблются вверх-вниз, то это свидетельствует об автокорреляции остатков, которая может быть оценена количественно с помощью критерия Дарбина-Уотсона. В этом случае метод наименьших квадратов неприменим, и необходимо использовать другие подходы.

Как было сказано выше, парная линейная регрессия не всегда хорошо описывает данные. Тогда зависимость можно попробовать смоделировать либо кусочно-линейной моделью, то есть построить несколько разных линейных моделей для разных диапазонов значений величины x, либо использовать нелинейные модели: гиперболическую, параболическую, степенную и обратную.

В общем случае при втором подходе оптимизационная задача формулируется так:

$$Q = \sum(f(x_i) - y_i)^2 \rightarrow min,$$

где $f(x_i)$ – некоторая нелинейная функция.

Для ее решения можно воспользоваться, например, методом Ньютона-Рафсона. Но в некоторых случаях при работе с парной нелинейной регрессией используют прием преобразования модели к линейному виду.

Рассмотрим способ преобразования более подробно. Например, мы выбрали гиперболическую модель, задаваемую уравнением вида

$$y = \beta_0 + \beta_1/x\ .$$

Тогда, если мы введем замену $z=1/x$, то модель преобразуется к привычному нам линейному виду.

Для параболической модели вида

$$y = \beta_0 + \beta_1 \times x + \beta_2 * x^2$$

замена не выполняется. При ее решении используется подход, аналогичный случаю линейной модели, но в итоговой системе будет три уравнения.

Обратная модель вида

$$y = 1/(\beta_0 + \beta_1 * x)$$

приводится к линейной через замену z=1/y.

Степенная модель вида

$$y = \beta_0 \times x^{\beta_1}$$

приводится к линейной логарифмированием:

$$\ln(y) = \ln(\beta_0) + \beta_1 \ln(x).$$

Вводя замены, получаем уравнение:

$$z = \beta_0{}' + \beta_1 \times t.$$

Однако, выполняя подобные преобразования, мы рискуем исказить взаимосвязи и понизить адекватность модели, поэтому линеризацию нужно применять крайне аккуратно. К тому же данный подход работает не для всех моделей. В таком случае, как было сказано выше, требуется использовать численный метод Ньютона-Рафсона. Опишем кратко его идею, не вдаваясь в подробности. Поиск решения в нем происходит посредствам построения последовательных приближений, то есть на основе простой итерации. Первым шагом задается стартовое приближение около предполагаемого корня. Затем в точке приближения строится касательная к графику функции, для которой находится пересечение с осью абсцисс. Эта точка считается следующим приближением, для которого повторяется процесс. Алгоритм продолжает свою работу до тех пор, пока не будет достигнута необходимая точность.

Мы рассмотрели различные модели парной регрессии, то есть случаи, когда отклик зависит от одного фактора. Однако в некоторых задачах на выходную переменную (отклик) влияет несколько факторов. В таком случае эта зависимость описывается множественной регрессией. Чаще всего в этом случае используются линейные модели. Общий вид зависимости описывается следующим образом:

$$y_i = \beta_0 + \beta_1 \times x_{1i} + \cdots + \beta_k \times x_{ki} + \varepsilon_i,$$

где $i=1...n$, n – количество наблюдений; k – количество факторов.

При работе с множественной регрессией, как правило, переходят к матричной записи системы:

$$Y=X\times\beta+\varepsilon,$$

где $Y = \begin{pmatrix} y_1 \\ ... \\ y_n \end{pmatrix}$ – вектор откликов; $\varepsilon = \begin{pmatrix} \varepsilon_1 \\ ... \\ \varepsilon_n \end{pmatrix}$ – вектор ошибок;

$\beta = \begin{pmatrix} \beta_1 \\ ... \\ \beta_n \end{pmatrix}$ – вектор параметров; $X = \begin{pmatrix} 1 & x_{11} \cdots & x_{k1} \\ 1 & x_{12} \cdots & x_{k2} \\ \cdots & \cdots \cdots & \cdots \\ 1 & x_{1n} \cdots & x_{kn} \end{pmatrix}$ – регрессионная матрица; n – количество наблюдений; k – количество факторов.

В данном случае задача формулируется следующим образом: для известных X и Y необходимо найти такие β, чтобы $Q=\sum \varepsilon_i^2 \rightarrow \min$.

В матричном виде последнее выражение можно переписать так:

$$Q = \varepsilon^T \varepsilon.$$

При этом

$$\varepsilon = Y - X\beta.$$

Тогда аналогом нормальной системы уравнений парной линейной регрессии в нашем случае будет следующее выражение:

$$\tilde{\beta} = (X^T X)^{-1} X^T Y.$$

Однако при работе с этой формулой могут возникнуть проблемы, связанные с тем, что обратную матрицу $(X^T X)^{-1}$ сложно посчитать. Как правило, это происходит, если ее столбцы сильно коррелированы. Например, если столбцы практически линейно зависят один от другого (имеет место мультиколлинеарность). Если коэффициент корреляции столбцов будет больше 0,8, то обратную матрицу уже не посчитать. Для решения этой проблемы либо исключают один из взаимозависимых факторов, либо прибегают к использованию гребневой регрессии (Ridge regression). Ее суть состоит в том, что для решения проблемы плохой обусловленности матрицы $(X^T X)$ к ней добавляется некое число λ. То есть обращается матрица $(X^T X + \lambda I)$. Полученные с помощью гребневой регрессии оценки параметров являются смещенными (в отличие от МНК-оценок), но доказано, что существует такое λ при котором оценки гребневой регрессии более эффективны, чем МНК-оценки. Однако четких рекомендация относительно выбора числа λ нет.

Продолжая тему проблем множественной регрессии, необходимо отметить еще одну. В случае непарной регрессии диаграмма рассеяния не применима для отслеживания выбросов, так как пространство будет многомерно. Для решения этой проблемы используют робастные методы. Основную информацию о них можно найти в источнике [14].

Если же говорить об общем алгоритме работы с множественной регрессией, то можно выделить следующие шаги:

1. Подготовка исходных данных, как правило, в виде таблицы (см. таблицу 8):

Таблица 8. Шаблон подготовки данных

№ опыта	y	x_1	x_2	…	x_k
1					
…					
n					

2. Оценивание параметров модели по формуле

$$\tilde{\beta} = (X^T X)^{-1} X^T Y,$$

или при использовании гребневой регрессии – по формуле

$$\tilde{\beta} = (X^T X + \lambda I)^{-1} X^T Y).$$

3. Проверка значимости модели с использованием критерия Фишера. Если по критерию Фишера линейная модель множественной регрессии оказалась незначима, то можно перейти к неполной квадратичной модели и проверить ее. Для двух факторов данная модель записывается следующим образом:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \beta_{12} x_1 x_2 .$$

Далее производят замену $\beta_3 = \beta_{12}$ и $x_3 = x_1 x_2$ и проверяют значимость получившейся модели. Если и она не значима, то переходят к полной квадратичной модели, затем (при ее незначимости) – к неполной кубической и полной кубической. Если же и последняя оказывается незначимой, то переходят к степенной, но здесь возрастает риск возникновения мультиколлинеарности. В принципе, даже кубическая модель уже дает высокую мультиколлинеарность. В случае, когда не удается найти ни одной значимой модели, признается, что регрессионными методами поставленную задачу решить нельзя, и ищется иной способ решения.

4. Проверка значимости каждого фактора по критерию Стьюдента. Если какой-то фактор оказывается незначимым, то его

убирают из расчета и повторяют его заново. Если незначимых факторов несколько, то убирают по одному.

5. Оценка качества модели с помощью коэффициента детерминации. В зависимости от области задачи приемлемы различные значения коэффициента детерминации. Например, для техники значение 0,9 будет пороговым. Все, что выше его, – хорошее, ниже – не очень. Для экономики порогом будет значение 0,5.

Теперь рассмотрим еще один особый вид регрессии – логистическую регрессию. Данный вид используется, если значение отклика (*y)* должно быть ограничено каким-либо диапазоном. Например, если *y* – вероятность некоторого события, то она должна лежать строго в диапазоне от 0 до 1. Рассмотренные ранее модели никак этого не учитывают, поэтому в подобных задачах используют логистическую регрессию. Чаще всего она применяется в задачах бинарной классификации. При $y < 0{,}5$ – один класс, иначе – другой.

В основе данного вида регрессии лежит следующая функция:

$$y = \frac{1}{1+e^{-x}}\ .$$

Ее график представлен на рисунке 14.

Соответствующая этой функции регрессионная модель будет иметь вид

$$y = \frac{1}{1+e^{-(\beta_0+\beta_1 x_1+\cdots+\beta_k x_k)}}\ .$$

Теоретически, выполнив преобразование линерализации, ее можно привести к линейной модели множественной регрессии, произведя следующую замену:

$$z = -\ln\left(\frac{1}{y-1}\right).$$

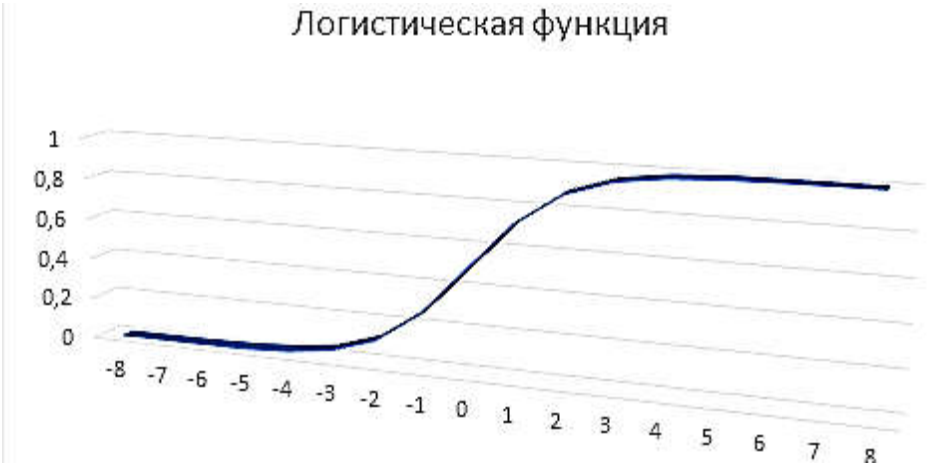


Рисунок 14. График логистической функции

Но так как в процессе преобразований нарушаются основные предпосылки МНК, то в практике этот подход не используется. Вместо этого для оценки параметров модели применяют метод максимального правдоподобия, с которым вам предлагается ознакомиться самостоятельно при необходимости.

Мы закончили рассмотрение статистических методов анализа данных. Теперь перейдем к более абстрактным моделям и алгоритмам, а именно к нечеткой логике.

Модели и методы нечеткой логики

Природа нечетких объектов обусловлена использованием экспертных оценок, которым свойственна неопределенность класса нечеткости. В отличие от стохастической неопределенности, нечеткость затрудняет или даже исключает применение статистических методов и моделей, но может быть использована для принятия предметно-ориентированных решений на основе приближенных рассуждений человека. Основные проблемы, решаемые в нечеткой логике, связаны с моделированием интеллектуальных операций приближенных рассуждений человека (эксперта), а также объектов, над которыми эти операции выполняются:

1. Объектами интеллектуальных операций, используемых в приближенных рассуждениях человека, являются переменные нового

класса – лингвистические переменные, значениями которых являются нечеткие множества. Важным является тот факт, что наименования лингвистической переменной и ее значений должны соответствовать словам, которые использует человек при решении прикладных задач. Таким образом, операндами и результатом интеллектуальных операций являются значения особого вида – *нечеткие множества*.

2. Основные интеллектуальные операции строятся с помощью *операций нечеткой логики*.

3. Алгоритмы вычисления нечетких значений предназначены для манипулирования со значениями, представленными нечеткими множествами на основе операций нечеткой логики, поэтому они классифицируются как нечеткие системы логического вывода. Часто используют сокращенную форму обозначенного класса моделей – *нечеткие модели или нечеткие системы.*

Нечеткие множества

Теория нечетких множеств, введенная Л. Заде [15], – это раздел прикладной математики, посвященный методам анализа неопределенных данных, в которых описание неопределенностей реальных явлений и процессов проводится с помощью понятия о множествах, не имеющих четких границ.

В дальнейшем для указания неопределенности экспертных оценок будем использовать эквивалентное выражение – лингвистическая неопределенность. Л. Заде предложил по аналогии с теорией вероятностей использовать в качестве математической модели лингвистической неопределенности объекта $x \in X$ функцию вида

$$Y = \mu\,(x,B),$$

где Y – результат вычисления функции, выражающий меру неопределенности (нечеткости) для конкретного объекта $x \in X$;

μ – непрерывная функция, такая, что $\mu: X \to [0,1]$. Содержательно функция μ определяет распределение неопределенности на X;

X – область определения функции μ. Область определения задается упорядоченным множеством значений произвольной природы, называемым *универсальным множеством (или универсумом*). Носителем функции $\mu(x,B)$ является подмножество $w \subset X$, на котором функция $\mu(x,B)$ принимает значение, отличное от нуля. В качестве универсального множества обычно задается множество действительных чисел;

B – вектор параметров функции, обычно числовых.

Функциональная модель лингвистической неопределенности получила название *нечеткого множества*, так как указанная функция μ рассматривается как характеристическая функция, определенная на множестве объектов X. Таким образом, с математической точки зрения, нечеткое множество моделируется параметрической функцией особого класса, называемого классом *функций принадлежности*. В том случае, если значения функции принадлежности нечеткого множества представлены точными числовыми значениями, такие нечеткие множества относят к *нечетким множествам типа 1*. Если значения функции принадлежности нечеткого множества моделируются другими нечеткими множествами, то такое нечеткое множество относят к *нечетким множествам типа 2*.

На практике используют несколько способов задания функции принадлежности. Среди них выделим следующие:

Структурный способ. Данная форма определения нечетких множеств основана на табличном представлении функций. В случае, если известен вектор параметров B, табличное представление функции принадлежности может быть задано явно путем табулирования функции $Y = \mu(x,B)$ на множестве значений w, являющемся ее носителем. При неизвестном векторе параметров B – путем прямого перечисления множества пар в виде

$$Y = \{\mu_1/x_1, \mu_2/x_2, .., \mu_n/x_n\}.$$

Данная форма удобна для графического отображения нечеткого множества и используется часто в тех случаях, когда затруднительно

задать математический вид функции $Y = \mu(x,B)$, например, если X не является множеством чисел.

Рассмотрим пример записи нечеткого множества в явной форме.

Пусть $w = \{x_1, x_2, x_3, x_4, x_5\}$, A – нечеткое множество, для которого $\mu_A(x_1) = 0{,}3; \mu_A(x_2) = 0; \mu_A(x_3) = 1; \mu_A(x_4) = 0{,}6; \mu_A(x_5) = 0{,}9$.

Тогда A можно представить в виде

$$A = \{0{,}3/x_1;\ 0/x_2;\ 1/x_3;\ 0{,}6/x_4;\ 0{,}9/x_5\}.$$

Функциональный способ. При этом предполагается, что форма функции принадлежности, моделирующей нечеткое множество, известна и определена на множестве действительных чисел X. Для представления $Y = \mu(x,B)$ используют различные функции (показанные, например, на рисунке 15: а) – треугольная, б) – трапецеидальная, в) – Гауссова).

Гауссова функция принадлежности описывается вектором параметров $B=\{\sigma, c\}$ и формулой

$$\mu(x, B) = \exp(-\left(\frac{x-c}{\sigma}\right)^2),$$

где c – среднее значение;

σ – среднее квадратичное отклонение.

Треугольная функция принадлежности характеризуется тройкой чисел, $B=\{a,b\ , c\}$, и вычисляется по формуле

$$\mu(x, B) = \begin{cases} \dfrac{x-a}{b-a}, a \le x \le b \\ \dfrac{c-x}{c-b}, b < x \le c \\ 0, x < a, x > c, \end{cases}$$

где b – задает координату вершины треугольника;

a, c – определяют основание треугольника.

По аналогии задается и трапецеидальная функция принадлежности, которая характеризуется четверкой чисел $B = \{a, b, c, d\}$:

$$\mu(x,B)=\begin{cases}\dfrac{x-a}{b-a}, a\le x\le b\\ 1, b<x<c\\ \dfrac{d-x}{d-c}, c\le x\le d\\ 0, x<a, x>d.\end{cases}$$

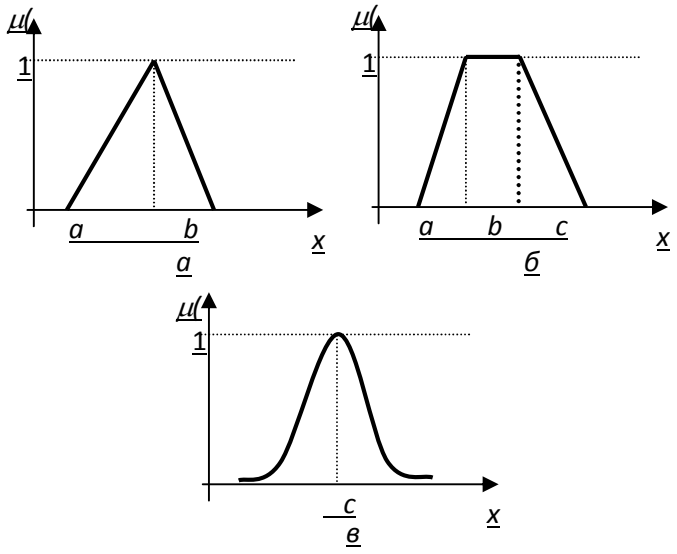


Рисунок 15. Типовые формы функций принадлежности

На практике часто параметр *B* явно не указывается для обеспечения более компактной записи функции принадлежности, то есть используется функциональная запись вида $Y=\mu(X)$ вместо $Y=\mu(x,B)$. При дальнейшем изложении в учебном пособии будем использовать компактную запись функции принадлежности.

С каждой функцией принадлежности $\mu(X)$ сопоставляется лингвистическое обозначение нечеткого множества (лингвистический терм). Тогда функция принадлежности нечеткого множества *Z* (функциональный способ) будет иметь следующую запись $Z=\mu_Z(X)$. Расширив традиционное понятие множества, Л. Заде построил «математический мостик» при описании свойств понятий в виде нечетких множеств между числом *x*, свойством *Z*, выраженным лингвистически, и степенью соответствия числа *x* свойству *Z* в виде функции

$\mu_Z(x)$. Фактически обозначение нечеткого множества через Z позволяет именовать функцию принадлежности $\mu_Z(X)$, в общем случае параметрическую, лингвистическими терминами, то есть оперировать с ней как со значениями лингвистической переменной. Часто эти два понятия – Z и $\mu_Z(X)$ – рассматриваются как эквивалентные.

Пусть $X = \{x\}$ – совокупность объектов (универсальное множество), обозначаемых через x. Пусть на X определены три нечетких множества: *A=«низкое», B=«удовлетворительное», C=«хорошее»,* обозначающие имена свойств понятия «качество». Тогда, следуя структурному способу для всех $x \in X$, нечеткое множество A может быть задано совокупностью упорядоченных пар $A = \{x, \mu_A(x)\}$, нечеткое множество B – совокупностью упорядоченных пар $B = \{x, \mu_B(x)\}$, нечеткое множество C – совокупностью упорядоченных пар $C = \{x, \mu_C(x)\}$. Используя функциональный способ записи, приведенный выше, нечеткие множества будут представлены следующим образом:

$$A = \mu_A(x),\ B = \mu_B(x),\ C = \mu_C(x).$$

Лингвистические переменные

Лингвистическая переменная – это переменная, значениями которой являются слова или высказывания естественного или искусственного языка.

Согласно [16]: «Поскольку слова в общем смысле менее точны, чем числа, понятие лингвистической переменной дает возможность приближенно описывать явления, которые настолько сложны, что не поддаются описанию в общепринятых количественных терминах… высокая точность несовместима с высокой сложностью. Таким образом, быть может, именно по этой причине обычные методы анализа систем и моделирования на ЭВМ, основанные на точной обработке численных данных, по существу не способны охватить огромную сложность процессов человеческого мышления и принятия решений. Отсюда напрашивается вывод о том, что для получения

существенных выводов о поведении гуманистических систем придется, по-видимому, отказаться от высоких стандартов точности и строгости, которые мы, как правило, ожидаем при математическом анализе четко определенных механистических систем, и относиться более терпимо к иным подходам, которые являются приближенными по своей природе».

Любая переменная описывается множеством допустимых значений, а лингвистические понятия описываются набором присущих им свойств. Л. Заде расширил понятие обычной лингвистической переменной, допустив, что в качестве ее значений (термов) выступают нечеткие переменные [16]. Пример лингвистической переменной, заимствованный из [17], представлен на рисунке 16.

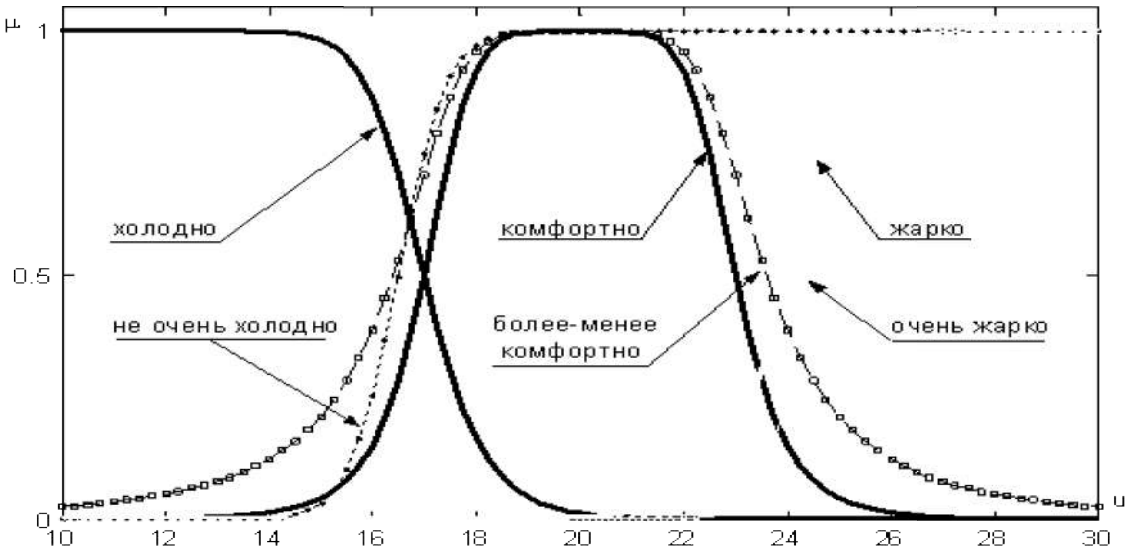


Рисунок 16. Пример лингвистической переменной «Температура»

Формально лингвистическая переменная описывается набором

$$< Name, \tilde{X}, X, G, P >,$$

где *Name*– наименование лингвистической переменной;

X – универсальное множество объектов x;

$\tilde{X}$ – базовое терм-множество, образующее совокупность термов лингвистической переменной, например, $\tilde{X}$ ={«Отличный», «Хороший», «Плохой», «Удовлетворительный» и др.};

G – синтаксические правила вывода (порождения) новых термов $\tilde{X}^*$, не входящих в базовое терм-множество, задаваемые обычно на основе контекстно-свободной грамматики;

P – семантические правила, контекстно-зависимый способ вычисления смысла на основе функций принадлежности каждого терма из $\tilde{X} \cup \tilde{X}^*$.

Операции нечеткой логики

Нечеткая логика – это логика, оперирующая нечеткими высказываниями и рассуждениями на базе частичной истинности.

В основе операций нечеткой логики лежит понятие нечеткого множества, выраженного функцией принадлежности. Поэтому операндами и результатами операций нечеткой логики являются также функции, определяющие новые нечеткие множества.

В нечеткой логике для моделирования основных логических связок И ($\wedge$), ИЛИ ($\vee$) над нечеткими множествами используют триангулярные нормы [18].

Триангулярной нормой (t-нормой) называют отображение $T:[0,1]\times[0,1]\to[0,1]$, удовлетворяющее следующим условиям:

T(0, 0)=0; T(x, 1)= x; T(1, x)= x – ограниченность;

$T(x,y)\le T(a,b)$, если $x\le a$, $y\le b$ – монотонность;

$T(x,y)=T(y,x)$ – коммутативность;

$T(x,T(y,z))\le T(T(x,y),z)$ – ассоциативность.

Триангулярной конормой (s-конормой) называют отображение $S:[0,1]\times[0,1]\to[0,1]$, удовлетворяющее следующим условиям:

S(1, 1)=1; S(x, 0)= x; S(0, x)= x – ограниченность;

$S(x,y)\ge S(a,b)$, если $x\ge a$, $y\ge b$ – монотонность;

$S(x,y)=S(y,x)$ – коммутативность;

$S(x,S(y,z))\le S(S(x,y),z)$ – ассоциативность.

t-норма и s-конорма в определенном смысле являются двойственными понятиями. Эти функции могут быть получены друг из

друга, например, с помощью инволютивного отрицания и законов Де Моргана следующим образом:

$$S(x,y) = n(T(n(x),n(y))),\ T(x,y) = n(S(n(x),n(y))).$$

Простейшими примерами t-норм и s-конорм, взаимно связанных этими соотношениями для *n(x)=1-x*, являются следующие (таблица 9).

Таблица 9. Примеры t-норм и s-конорм

Формула	**Интерпретация**
T(x, y) = min{x,y}	*(минимум)*
S(x, y) = max{x,y}	*(максимум)*
T(x, y) = xy	*(произведение)*
S(x, y) = x+y-xy	*(вероятностная сумма)*
T(x, y)=max{x+y-1, 0}	*(t-норма Лукасевича)*
S(x, y)=min{x+y,1}	*(t-конорма Лукасевича ограниченная сумма)*

Основные операции с нечеткими множествами

1. Операция эквивалентности

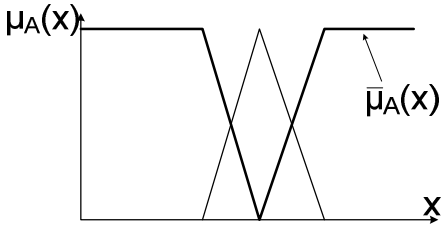
$$A \equiv B \Leftrightarrow \forall x \in X \ \ \mu_A(x) = \mu_B(x)$$

2. Операция включения

$$A \subseteq B \Leftrightarrow \mu_A(x) \le \mu_B(x), \forall x \in X$$

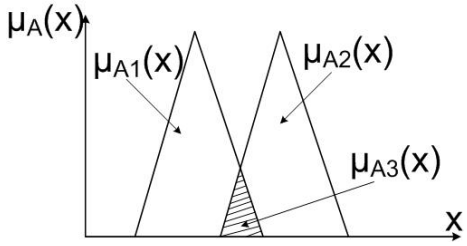
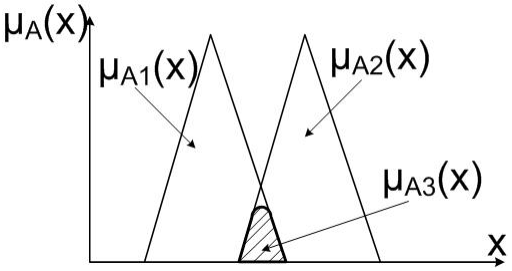
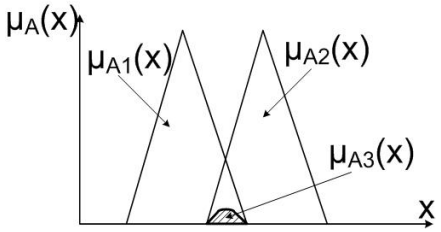
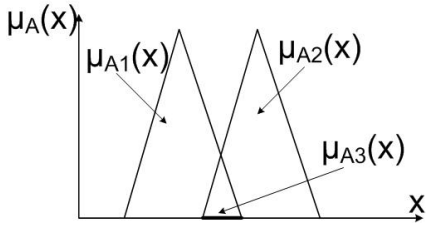
4. Нечеткая операция «НЕ» (дополнение) показана в таблице 10.

Таблица 10. Нечеткое «НЕ»

Иллюстрация	**Формула**
	$\overline{\mu_A}(x) = 1 - \mu_A(x)$ – нечеткая операция НЕ по Заде $\overline{\mu_A}(x) = \frac{1-\mu_A(x)}{1+\lambda \cdot \mu_A(x)}$ – НЕ по Сугено

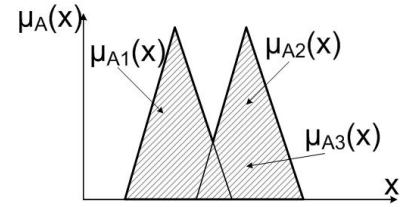
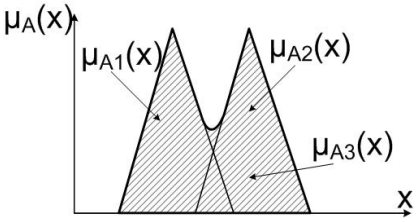
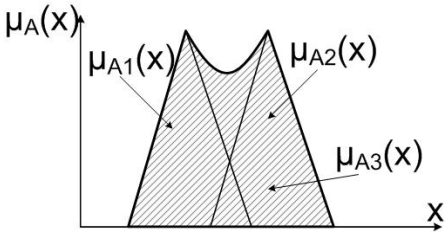
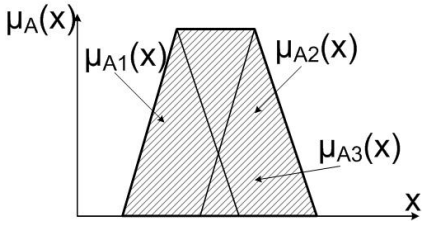
5. Нечеткая операция «И» показана в таблице 11.

Таблица 11. Нечеткое «И»

Иллюстрация	**Формула**
логическое произведение (Заде)	
	$\mu_{A_3}(x) = \mu_{A_1 \wedge A_2}$ $= \min(\mu_{A_1}(x), \mu_{A_2}(x))$ min – конъюкция
алгебраическое произведение (Бандлер, Коходт)	
	$\mu_{A_3}(x) = \mu_{A_1}(x) \cdot \mu_{A_2}(x)$
граничное произведение (Лукасевич, Гринс)	
	$\mu_{A_3}(x) = \mu_{A_1 \otimes A_2}$ $= \max(\mu_{A_1}(x) + \mu_{A_2}(x) - 1, 0)$
драстическое произведение (Вебер)	
	$\mu_{A_3}(x) = \mu_{A_1}(x) \triangle \mu_{A_2}(x) = \begin{cases} \mu_{A_1}(x), если\ \mu_{A_2}(x) \\ \mu_{A_2}(x), если\ \mu_{A_1}(x) \\ 0, иначе \end{cases}$ $0 \le \mu_{A_1 \triangle A_2} \le \mu_{A_1 \otimes A_2} \le \mu_{A_1 \cdot A_2} \le \mu_{A_1 \wedge A_2}$

6. Нечеткая операция «ИЛИ» показана в таблице 12.

Таблица 12. Нечеткое «ИЛИ»

Иллюстрация	**Формула**
логическая сумма (Заде)	
	$\mu_{A_3}(x) = \mu_{A_1 \vee A_2} = \max(\mu_{A_1}(x), \mu_{A_2}(x))$ max-дизъюнкция
алгебраическая сумма	
	$\mu_{A_3}(x) = \mu_{A_1}(x) + \mu_{A_2}(x) - \mu_{A_1}(x) \cdot \mu_{A_2}(x),$ $где\ \forall x \in X$
граничная сумма	
	$\mu_{A_3}(x) = \mu_{A_1 \oplus A_2} = \min(\mu_{A_1}(x) + \mu_{A_2}(x), 1)$
драстическая сумма	
	$\mu_{A_3}(x) =$ $\mu_{A_1}(x) \nabla \mu_{A_2}(x) = \begin{cases} \mu_{A_1}(x), если\ \mu_{A_2}(x) = 0 \\ \mu_{A_2}(x), если\ \mu_{A_1}(x) = 0 \\ 1, иначе \end{cases}$

По существу, все человеческие понятия являются нечеткими, так как они получаются в результате группировки (clumping) точек или объектов, объединяемых по сходству. Тогда нечеткость подобных

групп (clumps) есть прямое следствие нечеткости понятия сходства. Простыми примерами таких групп являются понятия «средний возраст», «деловая часть города», «немного облачно», «бестолковый» и др. Данную группу в нечеткой логике называют *«гранулой» (granule).* В естественном языке (ЕЯ) слова играют роль меток гранул и служат для сжатия данных. Сжатие данных с помощью слов является ключевым аспектом человеческих рассуждений и формирования понятий.

В нечеткой логике гранулирование информации лежит в основе понятий лингвистической переменной и нечетких правил типа «ЕСЛИ-ТО», задаваемой операцией импликации.

Операция импликации

В качестве основного математического инструмента при определении импликации A→B для нечетких множеств *A* и *B* используют композиционное правило Л. Заде [16], являющееся обобщением правила modusponens (таблица 13).

Таблица 13. Импликация

Описание	**Формула**
Предпосылка	$A \to B$
Событие	A^*
Вывод	$A^* \circ (A \to B)$

Пусть *U* и *V* – два универсальных множества с базовыми переменными *u* и *v* соответственно. Пусть *A* и *F* – нечеткие подмножества множеств *U* и *U×V*. Тогда композиционное правило вывода утверждает, что из нечетких множеств *A* и *F* следует нечеткое множество $B = A \circ F$. Функция принадлежности результата вычисляется с помощью триангулятных норм следующим образом:

$$\mu_B(v) = S(T(\mu_A(u), \mu_F(u,v)),$$

при моделировании s-конормы и t-нормы операциями (∨) *max* и (∧) *min* соответственно:

$$\mu_B(v) = \bigvee_{u \in U} ((\mu_A(u) \wedge \mu_F(u, v)) .$$

Другие формулы, реализующие операцию импликации, и математические объекты теории нечетких множеств и нечеткой логики приведены в учебном пособии [19].

Формализация нечеткой импликации позволила задать правила «ЕСЛИ-ТО» в виде нечетких продукционных правил и заложило основу нечеткого моделирования опыта и знаний экспертов, выраженных в виде приближенных зависимостей.

Нечеткие системы

Целью построения нечетких систем сложных явлений является приближенное описание зависимости (аппроксимация некоторой функции) $Y = f(X)$,

где Y – выходная лингвистическая переменная;

X – вектор входных лингвистических переменных. Его размерность – n;

f – зависимость между X и Y, описываемая совокупностью нечетких правил.

В основе нечетких систем лежат совокупность нечетких правил «ЕСЛИ-ТО», описывающих зависимости между нечеткими переменными предметной области, композиционное правило вывода и способ вычисления значений нечетких переменных (способ нечеткого вывода).

Системой нечеткого логического вывода в теории нечетких множеств и нечеткой логики называется модель, которая описывает поведение систем на естественном (или близком к естественному) языке в виде приближенных рассуждений на основе композиционного правила вывода.

В систему нечеткого логического вывода входят следующие объекты (см. рисунок 17):

- совокупность нечетких правил (база правил);
- набор функций принадлежностей базы нечетких переменных (база переменных);

- блок фаззификации;
- блок дефаззификации;
- блок вывода.

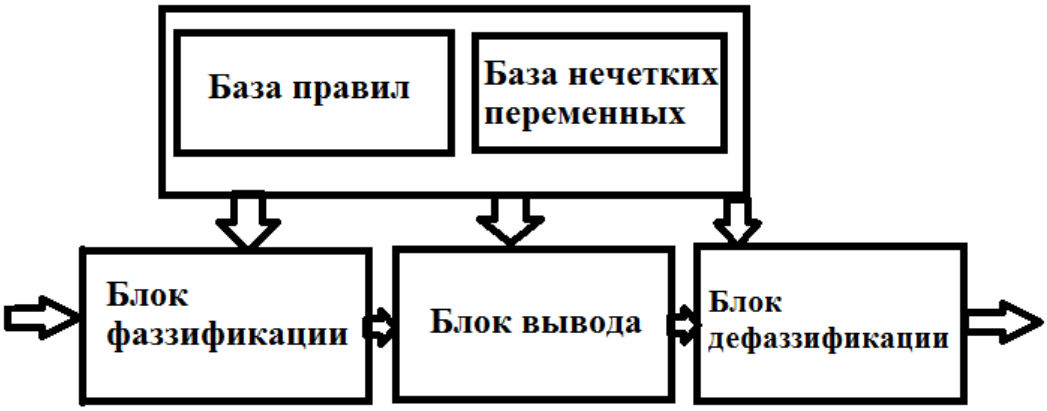


Рисунок 17. Система нечеткого вывода

База правил хранит множество логических правил вывода, а также их порядок (иерархическую структуру) применения. База нечетких переменных содержит названия лингвистических термов и параметры их функций принадлежности. База правил вместе с базой нечетких переменных образуют *базу знаний* (БЗ) системы нечеткого вывода.

Простейшие системы нечеткого логического вывода основаны на правилах вида:

R_i: *Если X есть* A_i *и Y есть* B_i, *то Z есть* C_i,

R_i: *Если X есть* A_i *и Y есть* B_i, *то* $z=f_i(x,y)$,

где X, Y – входные нечеткие переменные;

Z – выходная нечеткая переменная;

A_i, B_i, – входные значения (функции принадлежности);

C_i – выходные нечеткие значения (функции принадлежности);

f_i – некоторые вещественные функции.

При этом должны соблюдаться следующие условия:

- Существует хотя бы одно правило для каждого лингвистического терма выходной переменной.

- Для любого терма входной переменной имеется хотя бы одно правило, в котором этот терм используется в качестве предпосылки (левая часть правила).

В противном случае имеет место неполная база нечетких правил.

Распространены пять способов реализации нечеткого логического вывода [18].

Схема 1: Алгоритм Мамдани (Mamdani). Импликация моделируется минимумом, а агрегация – максимумом.

Схема 2: Алгоритм Цукамото (Tsukamoto). Исходные посылки – как у предыдущего алгоритма, но предполагается, что функции принадлежности являются монотонными.

Схема 3. Алгоритм Суджено (Sugeno). Алгоритм предполагает, что правые части правил вывода представлены в виде линейных функций.

Схема 4. Алгоритм Ларсена (Larsen). В алгоритме Ларсена нечеткая импликация моделируется с использованием операции умножения.

Схема 5. Упрощенный алгоритм нечеткого вывода. Исходные правила в данном случае задаются в виде

Если X есть A_i *и Y есть* B_i*, то* $z = Z_i$,

где Z_i – четкое значение.

Рассмотрим алгоритм нечеткого вывода по схеме Мамдани для базы правил вида R_i*: Если X есть* A_i *и Y есть* B_i*, то Z есть* C_i*.*, $i=[1,r]$.

1. Фаззификация.

Определяются степени истинности по функциям принадлежности для левых частей каждого правила:

$a_i = \mu_{A_i}(X)$

$b_i = \mu_{B_i}(Y)$,

где a_i – степень принадлежности X к A_i;

b_i – степень принадлежности Y к B_i;

$i= [1,r]$;

r – количество правил.

2. Импликация.

Определяется сила каждого правила, t-нормой является логический минимум:

$\alpha_i = \min(a_i, b_i)$.

Модифицируются функции принадлежности переменной z в каждом правиле:

$\mu'_i(z) = \min(\alpha_i, \mu_i(z))$,

где $\mu_i(z)$ – функция принадлежности переменной Z_i.

3. Агрегация.

Объединение выходов каждого правила логическим максимумом (s-конорма):

$\mu'(z) = \max_i(\mu_i(z))$

4. Деффазификация.

Простейшим способом выполнения процедуры дефаззификации является выбор четкого числа, соответствующего максимуму функции принадлежности. Однако пригодность этого способа ограничивается одноэкстремальными функциями принадлежности. Для многоэкстремальных функций на практике часто используется метод центра тяжести.

Дефаззификация нечеткого множества по методу центра тяжести осуществляется по формуле

$$x^0 = \frac{\int x \cdot \mu(x) dx}{\int \mu(x) dx}.$$

Физическим аналогом этой формулы является нахождение центра тяжести плоской фигуры, ограниченной осями координат и графиком функции принадлежности нечеткого множества. В случае дискретного универсального множества дефаззификация нечеткого множества по методу центра тяжести осуществляется по формуле

$$x^0 = \frac{\sum_{i=1}^{k} x_i \cdot \mu(x_i)}{\sum_{i=1}^{k} \mu(x_i)}.$$

На рисунке 18 графически представлен процесс нечеткого вывода по алгоритму Мамдани.

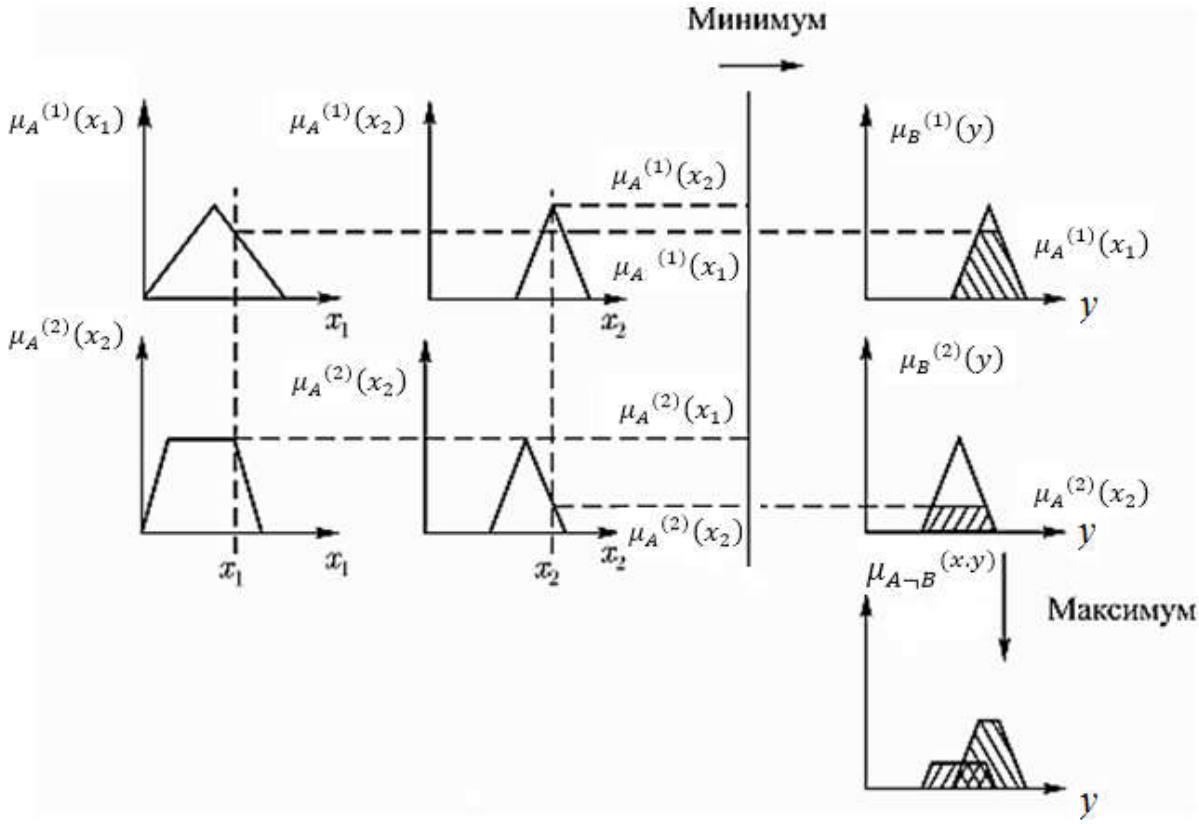


Рисунок 18. Алгоритм нечеткого вывода Мамдани

Пусть дана система управления с двумя правилами нечеткого управления:

Правило 1: IF x is A1 AND y is B1 THEN z is C1;

Правило 2: IF x is A2 AND y is B2 THEN z is C2.

Предположим, что величины x_0 и y_0, считываемые датчиком, являются четкими входными величинами для лингвистических переменных *x* и *y*, и что заданы следующие функции принадлежности для нечетких подмножеств *A1, A2, B1, B2, C1, C2* этих переменных:

$$\mu_{A1}(x) = \begin{cases} \frac{x-2}{3}, 2 \le x \le 5; \\ \frac{8-x}{3}, 5 \le x \le 8; \end{cases} \quad \mu_{A2}(x) = \begin{cases} \frac{x-3}{3}, 3 \le x \le 6; \\ \frac{9-x}{3}, 6 \le x \le 9; \end{cases}$$

$$\mu_{B1}(y) = \begin{cases} \frac{y-5}{3}, 5 \le y \le 8; \\ \frac{11-y}{3}, 8 \le y \le 11; \end{cases} \quad \mu_{B2}(x) = \begin{cases} \frac{y-4}{3}, 4 \le y \le 7; \\ \frac{10-y}{3}, 7 \le y \le 10; \end{cases}$$

$$\mu_{C1}(z) = \begin{cases} \frac{z-2}{3}, 1 \le z \le 4; \\ \frac{7-z}{3}, 4 \le z \le 7; \end{cases} \quad \mu_{C2}(z) = \begin{cases} \frac{z-3}{3}, 3 \le z \le 6; \\ \frac{9-z}{3}, 6 \le z \le 9; \end{cases}$$

Предположим, что в момент времени t_1 были считаны значения датчиков $x_0(t_1) = 4$ и $y_0(t_1) = 8$. Проиллюстрируем, как при этом будет вычисляться величина выходного сигнала.

Первым шагом находим α-срезы для первого и второго правила на основе заданных функций принадлежности (с учетом значений x_0 и y_0). С этой целью вычисляем величины функций принадлежности в заданных точках для первого и второго правил:

$$\mu_{A1}(x_0=4)=2/3 \quad \text{и} \quad \mu_{B1}(y_0=8)=1;$$
$$\mu_{A2}(x_0=4)=1/3 \quad \text{и} \quad \mu_{B2}(y_0=8)=2/3.$$

Затем, в соответствии с правилом вывода по Мамдани (выбор минимального значения функций принадлежности), определяем:

$$\alpha_1 = min(\mu_{A1}(x_0), \mu_{B1}(y_0)) = min(2/3, 1) = 2/3;$$
$$\alpha_2 = min(\mu_{A2}(x_0), \mu_{B2}(y_0)) = min(1/3, 2/3) = 1/3.$$

Результат применения вычисленных значений α_1 и α_2 показан на рисунке 19. Окончательный результат получается путем объединения найденных значений функций принадлежности с использованием оператора максимума (с учетом стратегии вывода по Мамдани). Результирующая функция принадлежности также представлена на рисунке 19.

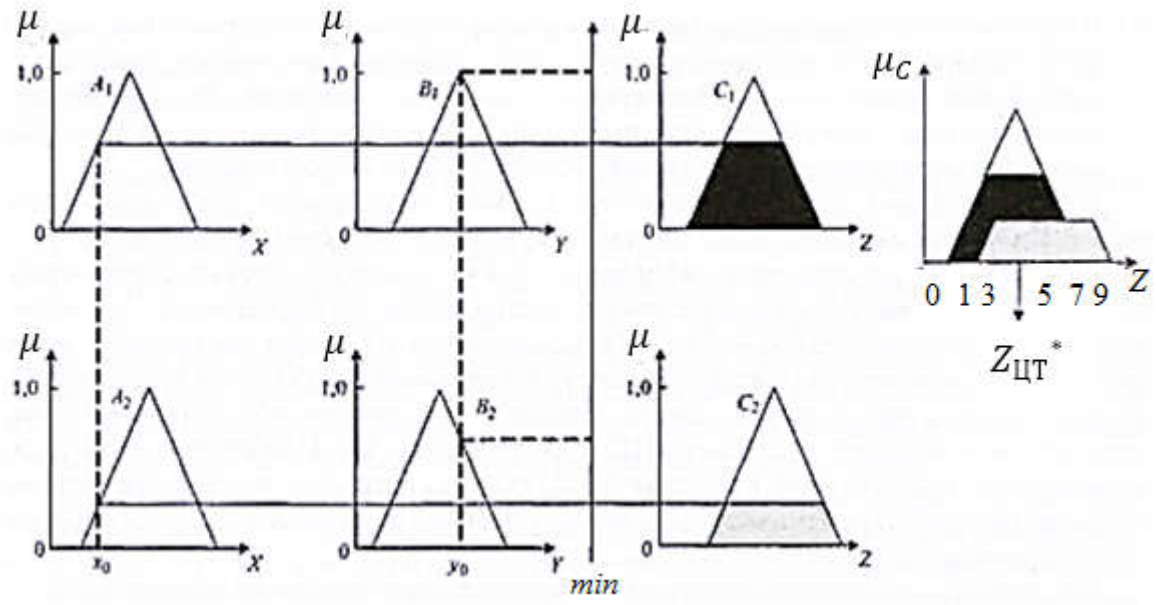


Рисунок 19. Иллюстрация нечеткого вывода по Мамдани

Для вычисления искомой выходной величины Z проводим дефаззификацию нечеткой величины μ_C. По методу центра тяжести получаем

$$Z_{\text{ЦТ}}{}^{*} = \frac{2\left(\frac{1}{3}\right) + 3\left(\frac{2}{3}\right) + 4\left(\frac{2}{3}\right) + 5\left(\frac{2}{3}\right) + 6\left(\frac{1}{3}\right) + 7\left(\frac{1}{3}\right) + 8\left(\frac{1}{3}\right)}{\left(\frac{1}{3}\right) + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right) + \left(\frac{2}{3}\right) + \left(\frac{1}{3}\right) + \left(\frac{1}{3}\right) + \left(\frac{1}{3}\right)} = 4,7.$$

Нечеткая логика в анализе временных рядов

Предположим, что задан процесс, состояния которого описываются n значениями одной переменной. Пусть в результате наблюдения получен временной ряд этой переменной, представляющий последовательность упорядоченных в равноотстоящие моменты времени пар $\{x_i, t_i\}$, таких, что $\forall x_i \in X, X \subset R^1, t_i \in N, i \in [1, n]$. Значение x_i – *уровень* временного ряда. Тогда введем понятие нечеткого временного ряда.

Нечетким временным рядом (НВР) называют упорядоченную в равноотстоящие моменты времени последовательность наблюдений над некоторым процессом, состояния которого изменяются во вре-

мени, если значение состояния процесса в момент t_i может быть выражено с помощью нечеткой метки $\tilde{x}_i$.

Нечеткая метка $\tilde{x}_i$ может быть сформирована непосредственно экспертом в форме

$$\mu_{\tilde{x}_i}(w, x_i),$$

где $\tilde{x}_i \in \tilde{X}$, $\tilde{X}$ – множество нечетких меток;

w – носитель (интервал на X) нечеткой метки $\tilde{x}_i$, $x_i \in w$;

$\mu_{\tilde{x}_i}(w, x_i) \in [0,1]$ – функция принадлежности нечеткой метки $\tilde{x}_i$ уровню временного ряда x_i, обычно треугольной формы.

Носитель нечеткой метки $\tilde{x}_i$ – это четкое множество $w \subseteq B$ таких точек $x_i \in w$, для которых $\mu_{\tilde{x}_i}(w) > 0$, где $B \subset X$ – базовое множество нечетких меток $\tilde{X}$.

Таким образом, нечеткий временной ряд формируется в результате интервального качественного оценивания уровней числового ВР. Интервалы-носители нечетких меток, образованные на множестве *X,* обязательно пересекаются. Качественный аспект нечеткой метке придает функция $\mu_{\tilde{x}_i}(w) \in [0,1]$.

На рисунке 20 изображен абстрактный нечеткий временной ряд, где каждой нечеткой метке $\tilde{x}_i$ соответствует нечеткое множество, задаваемое функцией принадлежности.

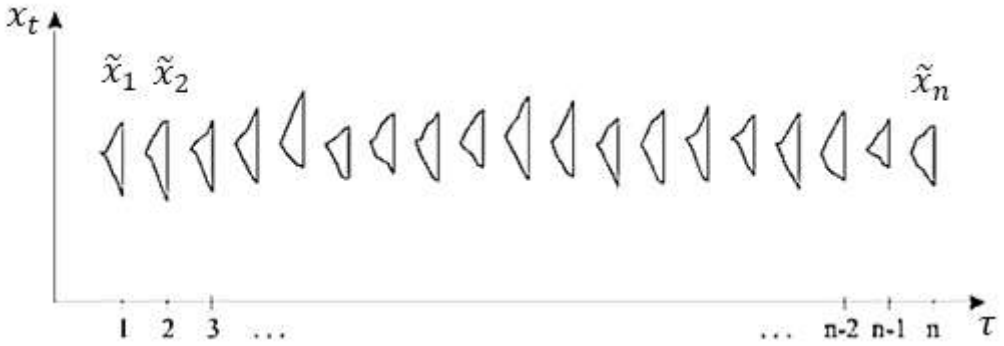


Рисунок 20. Абстрактный нечеткий временной ряд

В отличие от традиционного временного ряда значениями нечеткого ВР являются нечеткие множества, а не действительные значения уровней ВР.

В 1993 году ученые Сонг (Song) и Чиссом (Chissom) [20] предложили нечеткие модели детерминированных (time-variant) и авторегрессионных (time-invariant) временных рядов первого порядка (fist-order) и применили разработанные модели для прогнозирования количества регистрирующихся студентов университета штата Алабама (США), фаззифицировав предварительно четкий временной ряд. Это было первое применение нечетких моделей при моделировании ВР и первое определение моделей нечетких временных рядов.

Пусть $X_t,(t=1,...) \subset R^1$ – универсальное множество, на котором определены нечеткие множества $y_t^i,(i=1,2,...)$ и Y_t – коллекция $y_t^i,(i=1,2,...)$. Тогда Y_t называется нечетким ВР.

На практике в большинстве временных рядов последовательные наблюдения зависимы, так что:

$$R=\{(y_t,y_{t-1}),(y_{t-1},y_{t-2})...\} \subseteq Y_t \times Y_{t-1},$$

где Y_t, Y_{t-1} обозначают переменные;

y_t, y_{t-1} – наблюдаемые значения этих переменных.

Наиболее частой моделью зависимости является явная функция:

$$f: Y_{t-1} \to Y_t,$$

представленная линейной функцией:

$$y_t = f(y_{t-1},\phi,\varepsilon) = \phi y_{t-1} + \varepsilon_t,$$

где ε_t – случайная ошибка, шум.

В случае нечеткого временного ряда в качестве модели авторегрессии используется нечеткое разностное уравнение:

$$y_t^j = y_{t-1}^i \circ R_{ij}(t,t-1),$$

$$y_t^i \in Y_t,\ \ y_{t-1}^i \in Y_{t-1},\ \ i \in I,\ \ j \in J,$$

где ○ – обозначает операцию композиции из теории нечетких множеств;

$R(t,t-1)=\bigcup_{i,j} R_{ij}(t,t-1)$ – система нечетких отношений, которая символически может быть записана в виде $Y_t \to Y_{t-1}$.

Систему отношений R в выражении $Y_t = Y_{t-1} \circ R(t,t-1)$ называют *моделью нечеткого временного ряда* первого порядка. Данная модель – важный частный случай общей модели порядка p:

$$Y_t = (Y_{t-1} \times Y_{t-2} \times ... \times Y_{t-p}) \circ R(t,t-p),$$

$$R(t,t-p) = \max_{p} \left\{ \min_{j,i_1,i_2,..i_p} \left\{ y_t^j, y_{t-1}^{i_1}, ... y_{t-p}^{i_p} \right\} \right\}.$$

Метод моделирования нечетких временных рядов

Моделирование нечетких временных рядов в соответствии с нечеткой моделью, предложенной в работе [20], состоит в реализации следующих шагов:

1. Определение нечетких переменных – разбиение данных на множество интервалов (носителей нечетких множеств), определение лингвистических значений нечетких множеств и их функций принадлежности.
2. Формирование логических отношений $Y_t \to Y_{t-1}$.
3. Фаззификация входных данных – определение степени принадлежности входных данных входным нечетким переменным.
4. Вычисление результата применения нечеткого правила $R_{ij}(t,t-1)$ для каждой импликации.
5. Вычисление результирующего отношения R как объединения $\bigcup_{i,j} R_{ij}(t,t-1)$.
6. Применение полученной модели к входным данным и получение выходных нечетких результатов.
7. Дефаззификация нечетких результатов.

Одной из проблем в нечетком моделировании ВР является отсутствие четких рекомендаций на первом этапе построения модели

по выбору количества и параметров нечетких множеств, моделирующих входные и выходные переменные, в частности по определению их носителей (длины интервалов). Данные задачи выполняются экспертом, и, как показывают исследования, от выбора интервалов сильно зависит результат исследования.

Пример моделирования временного ряда в нечетком подходе

Приведем пример нечеткого моделирования временного ряда для прогнозирования прямых валютных котировок ЦБ USD/RUB за июнь 2005 года, описанный в работе [21] и представляющий модификацию метода Сонга, отличающуюся (а) использованием изменений (приращений) данных прошлого вместо реальных числовых значений (регистрации или валютного курса), и (б) вычислением отношений R_j для предсказания будущих состояний.

Рассматриваемый в работе метод был изначально успешно применен к временному ряду, характеризующему количество поступающих в Алабамский университет, которые являются бенчмаркингом при сравнении методов моделирования нечетких временных рядов.

Анализ результативности предлагаемого метода по показателю точности средней относительной ошибки аппроксимации для 6 нечетких множеств (MAPE=2,42) показал, что предложенный метод превышает аналогичный показатель для этого ВР, полученный методом Сонга и Чена (рисунок 21).

Применительно к проблеме прогнозирования валютного курса USD/RUB пошаговое описание предлагаемого метода нечеткого моделирования ВР можно свести к следующему:

Шаг 1: Задание области определения (универсального множества U) проблемы, исходя из вычисленных приращений валютного курса в течение рассматриваемого интервала времени.

Имеются следующие данные. Наибольшее положительное приращение курса доллара по отношению к российскому рублю наблю-

дается в феврале 2002 года, т. е. по сравнению с январским значением рост составляет 0.3679 (более 36 копеек/месяц). В ноябре-декабре 2004 года происходит самое значительное за трехлетний период наблюдений падение котировки доллара практически на 70 копеек (–0.6949). В результате, с целью упрощения последующего разбиения на равновеликие интервалы, полученные граничные значения (–0.6949 и +0.3679) слегка корректируются. Тогда, например, в случае использования шести подынтервалов U может быть представлена отрезком [–0.7, –0.5].

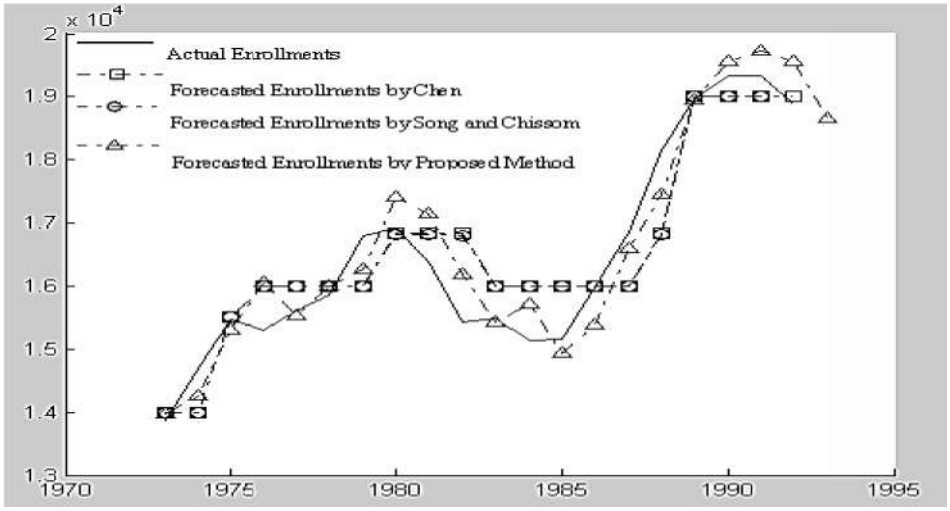


Рисунок 21. Сравнение результатов нечеткого моделирования

Шаг 2: Разбиение множества U на интервалы одинаковой длины.

Если мы оперируем с шестью нечеткими множествами, то область определения делится на 6 интервалов u_i, $i = \overline{1,6}$, $\mathrm{u}_1 = [-0.7, -0.5]$, $\mathrm{u}_2 = [-0.5, -0.3]$, ... , $\mathrm{u}_6 = [0.3, 0.5]$ (в действительности количество нечетких множеств не обязательно должно совпадать с числом интервалов разбиения).

Шаг 3: Определение нечетких множеств A_i.

Предположим, что лингвистическая переменная «изменение валютного курса» характеризуется терм-множеством, образуемым следующими значениями: A_1 (значительное уменьшение), A_2

(уменьшение), A_3 (без изменений/флэт), A_4 (увеличение), A_5 (значительное увеличение), A_6 (очень большое увеличение).

Для шести построенных выше интервалов u_i, $i = \overline{1,6}$, факт принадлежности каждого конкретного u_i определенному множеству A_j, $j = \overline{1,6}$ выражается действительным числом из единичного интервала [0,1] (предполагается, что элементы, отсутствующие в представлении множеств A_j, характеризуются нулевой степенью принадлежности):

$$A_1 = \{1/u_1 + 0.5/u_2\}$$
$$A_2 = \{0.5/u_1 + 1/u_2 + 0.5/u_3\}$$
$$A_3 = \{0.5/u_2 + 1/u_3 + 0.5/u_4\}$$
$$A_4 = \{0.5/u_3 + 1/u_4 + 0.5/u_5\}$$
$$A_5 = \{0.5/u_4 + 1/u_5 + 0.5/u_6\}$$
$$A_6 = \{0.5/u_5 + 1/u_6\},$$

где $u_i \subset U$ – элементы универсума U, а число, стоящее в числителе каждого элемента нечеткого множества, представляет собой степень принадлежности $\mu(u_i)$ этого элемента нечеткому множеству A_j, $j = \overline{1,6}$.

Шаг 4: Фаззификация приращений, полученных на шаге 1.

Считаем, что если приращение года t есть $p \in u_i$, и существует лингвистическое значение (нечеткое множество A_j) с максимальной степенью принадлежности, приходящейся на элемент u_i, тогда p фаззифицируется как A_j. Например, приращение за март 2002 по сравнению с предыдущим месяцем составляет +0.2476 – это значение попадает в интервал u_5, и фаззифицированное приращение становится равным A_5. Аналогичным образом производятся попарные сравнения каждого последующего и предыдущего месяцев, приводящие к формированию последовательности A_6 (февраль 2002), A_5

(март 2002), A_5 (апрель 2002), A_4 (май 2002), A_5 (июнь 2002), A_5 (июль 2002), A_4 (август 2002) и т. д.

Шаг 5: Формирование логических отношений $A_i \to A_j$.

Для построения последовательности логических отношений, рассматриваем попарно последовательные фаззифицированные приращения (февраль – март, март – апрель, и т. д.), определенные на шаге 4. Исключая повторяющиеся комбинации, окончательный список отношений принимает вид

$$A_1 \to A_2,\ A_1 \to A_4$$
$$A_2 \to A_1,\ A_2 \to A_2,\ A_2 \to A_3,\ A_2 \to A_4,\ A_2 \to A_5$$
$$A_3 \to A_2,\ A_3 \to A_3,\ A_3 \to A_4$$
$$A_4 \to A_2,\ A_4 \to A_3,\ A_4 \to A_4,\ A_4 \to A_5$$
$$A_5 \to A_3,\ A_5 \to A_4,\ A_5 \to A_5,\ A_5 \to A_6$$
$$A_6 \to A_5,\ A_6 \to A_4.$$

Мы предполагаем, что нечеткое импликативное отношение $D = B \to C$ для произвольных векторов B и C интерпретируется как нечеткая импликация Мамдани, следовательно, элементы матрицы D вычисляются по формуле $d_{ij} = b_i^T \times c_j = \min\left(b_i, c_j\right)$, где b_i и c_j – элементы векторов B и C, соответственно.

Шаг 6: Объединение логических отношений (шаг 5), имеющих одинаковые левые части, в группы, и вычисление отношений R_i, $i = \overline{1,6}$ для каждой сформированной группы. Можно обратить внимание на то, что группы отношений уже практически построены (см. шаг 5), и выглядят они следующим образом:

$$A_1 \to A_2,\ A_4$$
$$A_2 \to A_1,\ A_2,\ A_3,\ A_4,\ A_5$$
$$A_3 \to A_2,\ A_3,\ A_4$$
$$A_4 \to A_2,\ A_3,\ A_4,\ A_5$$
$$A_5 \to A_3,\ A_4,\ A_5,\ A_6$$
$$A_6 \to A_5,\ A_4.$$

Результирующие отношения $R_i, i=\overline{1,6}$, представляют собой объединения логических отношений, попавших в *i*-ю группу:

$$R_1 = A_1^T \times A_2 \cup A_1^T \times A_4$$
$$R_2 = A_2^T \times A_1 \cup A_2^T \times A_2 \cup A_2^T \times A_3 \cup A_2^T \times A_4 \cup A_2^T \times A_5$$
$$R_3 = A_3^T \times A_2 \cup A_3^T \times A_3 \cup A_3^T \times A_4$$
$$R_4 = A_4^T \times A_2 \cup A_4^T \times A_3 \cup A_4^T \times A_4 \cup A_4^T \times A_5$$
$$R_5 = A_5^T \times A_3 \cup A_5^T \times A_4 \cup A_5^T \times A_5 \cup A_5^T \times A_6$$
$$R_6 = A_6^T \times A_4 \cup A_6^T \times A_5.$$

Шаг 7: Прогнозирование и дефаззификация получаемых результатов.

Вычисленные отношения R_i используются в модели прогнозирования

$$A_i = A_{i-1} \circ R_i ,$$

где A_i – нечеткое множество, выражающее прогнозное приращение месяца i;

A_{i-1} – известное приращение предшествующего $(i-1)$-го месяца (если $A_{i-1} = A_j$, то $R_i = R_j$, $j=\overline{1,6}$);

$\circ$ – обозначает композиционный «max-min» оператор.

Например, приращение валютного курса за февраль 2004 года при известном приращении (–0.5714) за январь месяц того же года вычисляется по формуле $F(02.2004) = A_1 \circ R_1$, где R_1 имеет вид, показанный в первой строке, а A_i – фаззифицированное приращение января 2004 года.

Шаг 8: Вычисление прогнозных валютных котировок USD/RUB.

Этот этап предусматривает преобразование полученных на шаге 7 нечетких прогнозных приращений в целые числа. В значительной степени такой процесс зависит от особенностей рассматриваемой

задачи, и одним из критериев выбора процедуры дефаззификации является ее вычислительная простота.

После того как получено числовое приращение для рассматриваемого месяца, оно суммируется с уже имеющимся значением обменного курса предыдущего месяца. Рассмотренный метод нечеткого моделирования может быть отнесен к числу полуавтоматических процедур, поскольку большинство выполняемых шагов, включая построение универсума на основании множества исходных данных задачи, могут быть эффективно воплощены в программной форме. Однако участие аналитика (эксперта) при формировании интервалов разбиения и соответствующих нечетких множеств играет также огромную роль.

Извлечение знаний из временных рядов

Исследования временных рядов, в том числе нечетких, в последние десятилетия оформились в виде отдельного направления, называемого *интеллектуальным анализом данных,* или *DataMining*, в котором анализ временных рядов получил название *интеллектуального анализа временных рядов,* или *TimeSeries DataMining (TSDM).*

Человеческое знание основано на образах и формулируется лингвистически. Вычисления со словами и образами дают методологию для управления информацией и для разработки систем, основанных на знаниях. *DataMining* интегрирует методы, основанные на образах, дает возможность для извлечения информации из баз данных в лингвистической форме, подходящей для их использования в методах принятия решений. Методы и технологии извлечения знаний с использованием временных рядов должны оперировать паттернами временных рядов, отыскивать ассоциации между ними и извлекать знания (рисунок 22). То есть необходимо представление результатов *DataMining* в форме, используемой в человеческих знаниях.

На основе новой методологии *DataMining* решается расширенная совокупность задач анализа временных рядов, определенных в работе [22]:

1) сегментация – разбиение ВР на значимые сегменты;

2) кластеризация – поиск группировок ВР или их паттернов;

3) классификация – назначение ВР или их паттернам одного из заранее определенных классов;

4) индексирование – построение индексов для эффективного выполнения запросов к базам данных ВР;

5) резюмирование (summarization) – формирование краткого описания ВР, содержащего существенные черты с точки зрения решаемой задачи;

6) обнаружение аномалий – поиск новых, нетипичных паттернов ВР;

7) частотный анализ – поиск часто проявляющихся паттернов ВР;

8) прогнозирование – получение очередного значения ВР на основе истории ВР;

9) извлечение ассоциативных правил – поиск правил, относящихся к паттернам ВР.

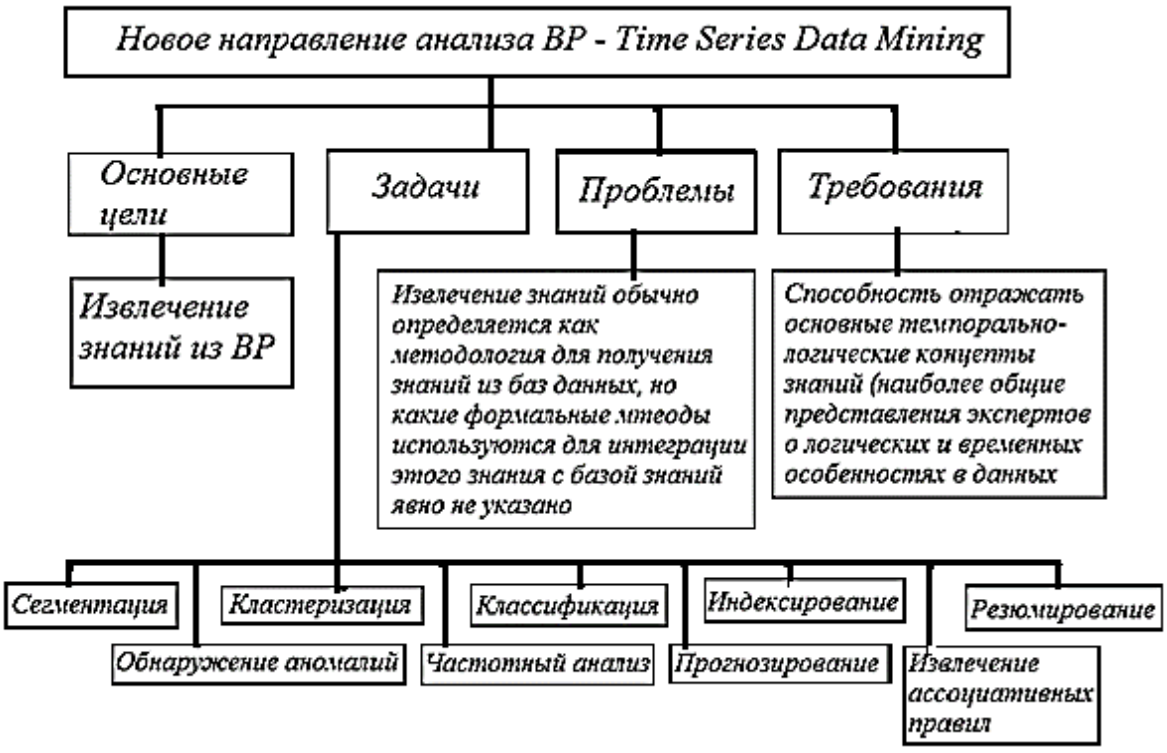


Рисунок 22. Задачи DataMining временных рядов

Традиционное выделение паттернов ВР было связано с выделением участков с постоянным знаком первой и второй производной: возрастающий и выпуклый, убывающий и гладкий и др. Различные шкалы и методы нечетких вычислений Л.Заде использовались для описания паттернов линейных трендов: рост, падение, резкий рост, медленное падение и т. д. Параметрические методы выпукло-гладкой модификации линейных функций и нечеткая грануляция выпукло-гладких паттернов позволили получить лингвистическое описание для ВР, подобное следующему: медленно убывающий и строго гладкий.

В рамках описанного выше направления TSDM акцент делается на поиск и извлечение правил из ВР, при этом полагаются на следующие основные принципы:

- Поиск правил нацелен на получение понимаемых результатов и не обязательно самых точных прогнозов;
- Важнейшим шагом на пути извлечения интерпретируемых знаний является порождение описаний фрагментов ВР в форме темпоральных образов, допускающих естественно-языковое толкование.

Требование к модели представления – способность отражать основные темпорально-логические концепты знаний (наиболее общие представления экспертов о логических и временных особенностях в данных).

Виды темпорально-логических концептов следующие:

- концепт временной продолжительности – присутствие определенного паттерна или признака ВР на определенном интервале времени;
- концепт очередности – порядок следования паттернов ВР во времени;
- концепт одновременности – совпадение во времени темпоральных событий (паттернов различных ВР);

- концепт нечеткости – нечеткость выраженности темпоральных событий и отношений.

В качестве инструментов анализа предлагается использовать нечеткие нейронные сети.

Существует множество методов прогнозирования временного ряда. Однако не создан лучший в любой ситуации метод: каждый из них имеет свои достоинства и недостатки. Поэтому одним из наиболее перспективных научных направлений стало создание комбинированных моделей.

Комбинированная модель прогнозирования – модель прогнозирования, состоящая из нескольких индивидуальных (частных) моделей, называемых базовым набором моделей. Использование комбинированной модели может повысить точность получаемого прогноза по ряду причин:

- Попытка выбрать одну модель для временного ряда с изменяющимся уровнем и динамическими свойствами приводит к выбору усредненной модели;
- При быстром изменении уровней ряда и его динамических свойств невозможно быстро производить анализ динамики и заменять одну модель прогнозирования другой;
- Любой отвергнутый из-за неоптимальности прогноз почти всегда содержит полезную независимую информацию;
- Слабые стороны одного метода прогнозирования возможно преодолеть, используя преимущества другого метода.

На кафедре «Информационные системы» Ульяновского государственного технического университета была разработана информационная система «Combination of fuzzy and exponential models». Данная система позволяет получать прогноз для временного ряда путем агрегации индивидуальных прогнозов моделей из базового набора. На момент создания системы базовый набор содержал 29 индивидуальных моделей и их модификаций, относящихся к экспоненциальным и нечетким моделям прогнозирования временных

рядов. Разработанная информационная система одержала победу на конкурсе «Computational Intelligence in Forecasting» в 2015 году [23].

Точность агрегированного прогноза напрямую зависит от выбранных из базового набора моделей. В связи с этим можно утверждать, что информационная система «Combination of fuzzy and exponential models» имеет два существенных недостатка:

- Пользователь вынужден самостоятельно выбирать модели из базового набора для каждого прогнозируемого временного ряда;
- Для всех временных рядов используется один агрегирующий метод.

Таким образом, целесообразно использовать методы машинного обучения для выбора моделей из базового набора и для выбора агрегирующего метода. Указанные задачи можно решить с помощью нейронной сети, выбирающей методы на основании значений метрик прогнозируемого временного ряда.

Для выбора метрик необходимо выбрать совокупность значимых для прогнозирования временного ряда характеристик, таких как: длина, степень выраженности тренда, степень выраженности сезонности, величина дисперсии, наличие стационарности.

При выборе используемых метрик временного ряда необходимо выполнение следующих условий:

- Метрика соответствует одной из основных характеристик временного ряда;
- Совокупность метрик максимально и разносторонне опиисывает временной ряд;
- Значение метрики принадлежит интервалу от 0 до 1.

При выборе методов прогнозирования из базового набора обучающая выборка нейронной сети представляет собой набор временных рядов, представленных в виде совокупности метрик, и соответствующих значений ошибки прогнозирования для каждой модели. Нейронная сеть, таким образом, обучается на основании значений метрик вычислять предполагаемые значения ошибки для каждой

модели из базового набора, что позволяет определять, какие модели из базового набора эффективнее применить для прогнозирования значений текущего временного ряда.

Для создания комбинированной модели прогнозирования на основании использования методов машинного обучения необходимо решить ряд задач:

- Определить состав моделей, входящих в базовый набор;
- Выявить ключевые для задачи прогнозирования метрики временного ряда;
- Определить структуру и способ обучения нейронных сетей выбора методов прогнозирования из базового набора и выбора метода агрегации.

Нечеткое сглаживание временного ряда

Очень часто при анализе временных рядов переходят от исследования самих значений ряда к исследованию тренда. Одним из методов его выделения является метод F-преобразования.

Нечеткое сглаживание временных рядов на основе нечеткого преобразования (F-преобразования) – методика, разработанная Перфильевой [24], которая может быть отнесена к методикам нечеткого приближения.

F-преобразование предполагает задание нечеткого разбиения универсального множества. В качестве последнего выбирается конечный интервал [a,b] действительной прямой. Зафиксируем значение n $(n > 2)$ узлов $x_1,...,x_n$ на [a,b] и предположим, что $x_1 \prec ... \prec x_n$, причем $a = x_1, b = x_n$.

Под нечетким разбиением [a,b] будем понимать совокупность n функций $A_1,...,A_n : [a,b] \to [0,1]$, удовлетворяющих следующим свойствам:

- $A_k : [a,b] \to [0,1], A_k(x_k) = 1$;

- $A_k(x) = 0$ если $x \notin (x_{k-1}, x_{k+1})$, где для единообразия обозначения мы положим $x_0 = a, x_{n+1} = b$;
- $A_k(x)$ непрерывна;
- $A_k(x), k = 2,...,n$ строго возрастает на $[x_{k-1}, x_k]$ и строго убывает на $[x_k, x_{k+1}]$;
- $\sum A_k(x) = 1$ для всех $x \in [a,b]$.

Функции $A_1,...,A_n$ называются базисными функциями. Базисные функции $A_1,...,A_n$ могут служить также функциями принадлежности нечетких подмножеств $A_1,...,A_n$ (обозначения функций и множеств унифицированы). Отметим, что форма базисных функций может быть уточнена дополнительно и согласована с такими требованиями к модели, как, например, гладкость.

Следующие формулы представляют нечеткое разбиение отрезка $[x_1, x_n]$, полученное совокупностью функций:

$$A_1(x) = \begin{cases} 1 - \dfrac{(x - x_1)}{h_1}, x \in [x_1, x_2] \\ 0, иначе \end{cases}$$

$$A_k(x) = \begin{cases} \dfrac{(x - x_k)}{h_{k-1}}, x \in [x_{k-1}, x_k], \\ 1 - \dfrac{(x - x_k)}{h_k}, x \in [x_k, x_{k+1}], \\ 0, иначе \end{cases}$$

$$A_{n-1}(x) = \begin{cases} 1 - \dfrac{(x - x_{n-1})}{h_{n-1}}, x \in [x_{n-1}, x_n] \\ 0, иначе \end{cases}$$

где $k = 1,...,n-1$ и $h_k = x_{k+1} - x_k$.

Предположим, что функция *f* имеет своей областью определения множество $P = \{p_1,..., p_l\} \subset [a,b]$, где $l>n$. Множество P считается плотным относительно нечеткого разбиения $A_1,...,A_n$, если выполнено условие:

$$(\forall k)(\forall j) A_k(p_j) > 0 \quad .$$

Пусть $A_k(p_j) = a_{kj}, k = 1,...,n; j = 1,...,l$, тогда матрица $A_{n\times l} = a_{kj}$ называется матрицей нечеткого разбиения для *P*, для которой справедливы свойства:

$$(\forall k)(\forall j) a_{kj} \in [0,1]$$

$$(\forall j)\sum_{k=1}^{n} a_{kj} = 1 .$$

F-преобразованием вектора *f*, определяемым матрицей нечеткого разбиения *A*, назовем вектор $F_n[f]$, где $F_n[f] = (F_1...F_n)$ и $F_i = \dfrac{\sum_{j=1}^{l} a_{ij} f_j}{\sum_{i=1}^{l} a_{ij}}$.

Координаты вектора $F_n[f]$ назовем соответственно компонентами F-преобразования. Обозначим $a_i = \sum_{j=1}^{l} a_{ij}, i = 1,..., n$; тогда $(a_1F_1,..., a_nF_n)^T = A \times f$. Компоненты F-преобразования являются точками минимума функции, задающей критерий взвешенного среднеквадратичного отклонения.

Пусть $F_n[f]$ есть F-преобразование *f*, определяемое $A_{n\times l} = a_{kj}$. Обратным F-преобразованием $F_n[f]$ назовем вектор $f_{F,n}$, вычисляемый по формуле $f_{F,n}^T = F_n[f] \times A$. Можно доказать, что если *n* возрастает, тогда $f_{F,n}(p_j)$ сходится $f(p_j), j = 1,..., N$.

F-преобразование имеет (кроме прочих) следующие свойства, важные для использования в качестве сглаживания временных рядов:

- У него прекрасные фильтрующие свойства;
- Его легко вычислять
- F-преобразование стабильно относительно выбора точек $p_1,...,p_N$. Это означает, что при выборе других точек p_k (и, возможно, изменяя их число *N*), результирующая функция $f_{F,n}$ значительно не меняется.

Прогнозирование тренда и прогнозирование числового представления временного ряда производится раздельно. Для этого необходимо вычислить так называемые остатки – разность между временным рядом и его трендом:

$$R = \{(f(p_j) - F_k)\}, \text{где } j : A_k(j) > 0 .$$

Полученный вектор *R*-вектор остатков для *k*-й компоненты тренда. Прогноз тренда и векторов остатков реализуется по формуле линейной комбинации. Прогноз компоненты тренда:

$$F_{k+1} = \alpha F_k + \beta F_{k-1}...$$

и прогноз вектора остатков:

$$R_{k+1} = \alpha R_k + \beta R_{k-1}... .$$

Для получения прогноза находится решение системы уравнений:

$$\begin{cases} F_{k-1} = \alpha F_{k-2} + \beta F_{k-3}... \\ F_k = \alpha F_{k-1} + \beta F_{k-2}... \end{cases}$$

Аналогично строится прогноз вектора остатков.

Также для построения прогноза может быть использована нейронная сеть, например, многослойный перцептрон или сеть Кохонена с выходной звездой Гроссберга. Вид нейронной сети определяет вид входных данных: на вход многослойного перцептрона подаются только абсолютные значения ($F_k, F_{k-1,...}$), на вход сети Кохонена подаются значения точек тренда, вычисленные относительно друг друга ($F_k - F_{k-1}, F_{k-1} - F_{k-2},....$).

На качество прогноза влияет количество нейронов во внутреннем слое. После получения прогнозных значений тренда и вектора остатков возможно построение числового представления прогноза временного ряда. Для этого производится сложение прогнозной компоненты тренда и прогнозного вектора остатков.

Точность прогноза зависит от количества точек временного ряда, участвующих в обучении: чем больше их, тем меньше ошибка (рисунок 23). Как видно из рисунка, вывод справедлив и для MAPE, и для SMAPE (линии на графике практически совпадают).

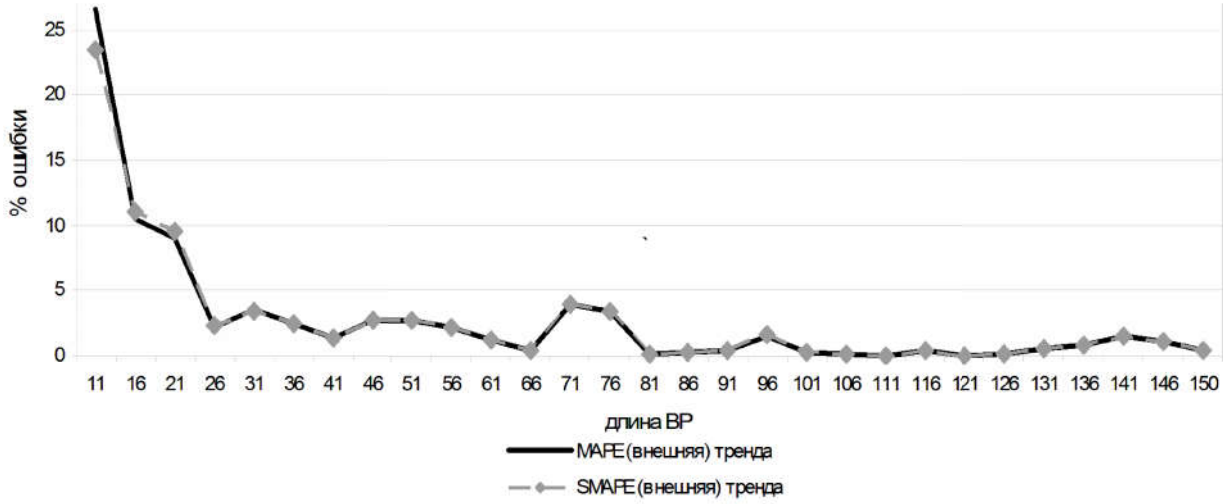


Рисунок 23. Зависимость процента ошибок от длины обучающей выборки

Процесс получения прогноза тренда и остатков зависит от некоторых параметров, которые в модели не удается задать жестко. Выбор данных параметров в значительной степени влияет на качество прогноза. Определим их:

- Степень авторегрессии при построении прогноза тренда (область значений);
- Метод, которым производится прогноз:
  - решение системы линейных уравнений;
  - нейронная сеть с абсолютными значениями предыдущих компонент на входе;
  - нейронная сеть с разностями между предыдущими компонентами на входе;

- нейронная сеть с абсолютными значениями предыдущих компонент, а также разности между ними на входе;
- количество точек, покрываемых базисной функцией;
- сезонность (история отстоит на k точек от прогноза).

Сочетание параметров, при котором получается наилучший прогноз, получается перебором.

Нечеткая регрессия

В области прикладной статистики, анализа временных рядов и принятия решений в условиях неопределенности накоплен богатый опыт исследований и существует множество моделей, начиная от простейших линейных регрессионных моделей поиска тренда временного ряда и заканчивая сложными многоуровневыми авторегрессионными и адаптационными моделями. Регрессионный анализ, основанный на методе наименьших квадратов (Least-square), является очень удобным методом построения моделей, позволяющим численно оценивать зависимость интересующего исследователя параметра от воздействующих на него факторов. При анализе зависимости нечетких оценок от воздействующих факторов зачастую исследователям приходится иметь дело с важной информацией, которая не может быть задана точно. Некоторые наблюдения могут быть описаны только лингвистическими выражениями (типа «удовлетворительный», «хороший» и «превосходный»). Для таких данных аппаратом формализации может служить теория нечетких множеств. Были разработаны различные нечеткие регрессионные модели, основой которых является модель нечеткой линейной регрессии.

В нечеткой регрессионной модели параметры представляются триангулярными нечеткими числами и являются коэффициентами в нечеткой линейной функции. Неопределенность (vagueness) системы представляется суммарным разбросом («шириной») параметров (нечетких коэффициентов).

Построение модели состоит в нахождении оптимальных в некотором смысле коэффициентов с учетом нечеткой информации об объекте и субъективных представлений исследователя.

Базовые предположения нечеткой регрессии заключаются в том, что остатки, полученные как разность между наблюдениями и их оценками, продуцируются не случайными ошибками измерения, а неопределенностями (типа нечеткость) при вычислении параметров модели.

Большинство работ, посвященных нечеткой регрессии, были основаны на следующих базовых определениях.

Пусть дано множество наблюдений: $(y_j, x_{j1}, ...x_{jn}), j = 1,..., m$, необходимо найти нечеткую модель по следующей форме:

$$\tilde{Y} = A_0^{\sim} + A_1^{\sim} x_1 + ... + A_n^{\sim} x_n,$$

где $A_i(a_i^c, s_i^L, s_i^R),\ i = 1,..., n$ – триангулярные нечеткие числа;

a_i^c – среднее значение $A_i^{\sim}$;

s_i^L, s_i^R – показывают левый и правый разброс треугольной функции принадлежности соответственно.

Используются два критерия определения нечетких коэффициентов модели:

1. Для всех наблюдений принадлежность значения y_j к его нечеткой оценке $Y_j^{\sim}$ должна быть как минимум $Y_j^{\sim}(y_j) \geq h, j = 1,..., m$, где h – уровень доверия, выбранный лицом, принимающим решения.
2. Общая нечеткость предсказываемого значения зависимой переменной должна быть минимизирована. Это может быть достигнуто минимизацией суммы разбросов нечетких чисел для всех наборов данных.

Итак, проблему настройки нечеткой модели с заданными данными $(y_j, x_{j1},...x_{jn}), j = 1,..., m$ можно решить как эквивалентную задачу линейного программирования. То есть найти $a_-^c = (a_0^c,...a_n^c)$, $s_-^L = (s_0^L,...s_n^L)$, $s_-^R = (s_0^R,...s_n^R)$, которые минимизируют выражение:

$$Z = m(s_0^L + s_0^R)s_0^L + \sum_{I=1}^{n}\left[\left(s_i^L + s_i^R\right)\sum_{j=1}^{m}\left|x_{ij}\right|\right].$$

Чтобы оценить качество настройки нечеткой регрессии, используют метод наименьших квадратов. Для нечеткой регрессии среднеквадратичное отклонение (MSE) определяется следующим образом:

$$MSE = \frac{1}{m}\sum_{j=1}^{n}[y_j - def(Y_j)]^2 ,$$

где $def(Y_j)$ – дефаззифицированное значение зависимой переменной.

Исследователями были выделены различные варианты методов на основе классификации «вход – выход»: «четкий вход – четкий выход» метод CICO (Crisp-Input and CrispOutput), «нечеткий вход – нечеткий выход» метод FIFO (Fuzzy-Inputs and Fuzzy-Outputs) и смешанные данные – метод CIFO (Crisp-Inputs and Fuzzy-Outputs).

ACL-шкала и нечеткая кластеризация объектов

Для задания отношений между элементами нечеткого временного ряда мы можем использовать специальную лингвистическую шкалу в качестве инструмента как абсолютного, так и сравнительного нечеткого оценивания – ACL-шкала (Absolute&Comparative Linguistic) [25]. Абсолютные оценки, полученные по ACL-шкале, будут соответствовать нечетким меткам уровней ВР, а сравнительные оценки – нечетким тенденциям НВР. Опишем ACL-шкалу [25] как алгебраическую систему вида

$$Sx = <Name\_Sx, \tilde{X}, N, X, G, P, TTend, RTend>,$$

где *Name_ Sx* – имя ACL-шкалы; $\tilde{X}$ – базовое терм-множество абсолютных лингвистических оценок (лингвистическое название градаций); *N* – мощность базового терм-множества шкалы; *X* – универсальное множество, на котором определена шкала; *G* – синтаксические правила вывода (порождения) цепочек оценочных высказываний (производные термов, не входящих в базовое терм-множество); *P* – семантические правила, определяющие функции принадлежности для каждого терма (задаются обычно экспертно); $TTend(\tilde{x}_i,\tilde{x}_j)$ – лингвистическое отношение, фиксирующее тип изменения между двумя оценками $\tilde{x}_i,\tilde{x}_j$ шкалы; $RTend(\tilde{x}_i,\tilde{x}_j)$ – лингвистическое отношение, фиксирующее интенсивность различия между двумя оценками $\tilde{x}_i,\tilde{x}_j$ шкалы.

Первое, что нужно сделать для построения шкалы, – определить базовое терм-множество. Например, определим множество мощностью 5, содержащее в себе оценки уровней тренда исходного временного ряда: {малое, ниже среднего, среднее, выше среднего, большое}. Мощность множества обуславливается необходимостью вербализации каждого понятия шкалы, таким образом, чтобы она была понятна для человека и семантически содержательна.

Далее необходимо выбрать способ построения ACL-шкалы. Этим способом может стать неравномерное разбиение зафиксированной области значения величины с помощью алгоритма кластеризации. Достоинством такого подхода является то, что разбиение происходит без участия эксперта (тогда как в остальных возможных способах обязательно участие эксперта).

При безэкспертном разбиении задача формулируется следующим образом: дано множество из *L* точек, необходимо разбить его на *k* групп. Здесь *L* – количество элементов ВР (зафиксированное множество значений), *k* – количество термов на ACL-шкале, в нашем случае равное *N*. Как мы видим по формулировке задачи, это задача кластеризации. Кластеризация – это процесс разбиения объектов на группы по степени их схожести между собой. В отличие от

классификации, где есть заданная структура групп и признаки, на основе которых объекты в эти группы помещаются, в кластеризации структура разбиения, а также характеристические признаки являются не входными, а выходными данными. Кластеризация может быть четкой и нечеткой. При четкой кластеризации предполагается, что каждый объект принадлежит только одному кластеру, при нечеткой объект принадлежит всем кластерам, но с разной степенью принадлежности. При нечеткой кластеризации мы сможем использовать собственно степени принадлежности объектов ВР кластерам для получения термов шкалы. В итоге вся зафиксированная область разобьется на N взаимопересекающихся кластеров, упорядоченных по возрастанию значений их центров. Взаимопересечение получается за счет нечеткости кластеризации.

Кластеризация может выполняться с помощью нейронных сетей, генетических алгоритмов или с помощью собственно алгоритмов кластеризации. Например, для нечеткого разбиения некого множества точек на заданное количество кластеров можно использовать fcm-алгоритм кластеризации. Рассмотрим его подробнее.

FCM-алгоритм (Fuzzy Classifier Means) кластеризации применяется для нечеткого разбиения некого множества объектов на заданное количество кластеров. Целью алгоритма кластеризации является автоматическое разбиение множества объектов, которые задаются векторами признаков в пространстве признаков. Этот алгоритм предполагает, что объекты принадлежат всем кластерам с определенной степенью принадлежности, которая определяется расстоянием от объекта до соответствующих кластерных центров. Данный алгоритм итерационно вычисляет центры кластеров и новые степени принадлежности объектов. При этом он основан на минимизации целевой функции по мере расстояния объектов от центров кластеров:

$$J = \sum_{i=1}^{N} \sum_{j=1}^{C} (u_{ij})^m \left\| x_i - c_j \right\| \quad ,$$

где N – количество объектов; C – количество кластеров; u_{ij} – степень принадлежности объекта i кластеру j; m – любое действительное число, большее 1; x_i– i-й объект набора объектов; c_j– j-й кластер набора кластеров; $||x_i - c_j||$ – норма, характеризующая расстояние от центра кластера j до объекта i.

Объектами кластеризации при анализе текстов, например, могут являться термины. Вектор признаков объектов кластеризации в данном случае содержит только значение какой-либо статистической метрики (например, частоты термина в тексте).

Алгоритм нечеткой кластеризации выполняется по шагам. Рассмотрим каждый из шагов подробнее, приведя необходимые модификации целевой функции для упрощения дальнейшего программирования.

Первый шаг – инициализация. На этом шаге задаются параметры кластеризации и инициализируется первоначальная матрица принадлежности объектов кластерам. К параметрам относятся следующие величины. Во-первых, степень нечеткости кластеризации – параметр m. От выбора этого параметра зависит значение функции принадлежности точки кластеру. Обычно он выбирается в пределах ~1.5..2. Чем больше его значение, тем более «нечеткая» кластеризация. В приведенных ниже примерах кода эмпирически было выбрано значение = 1.6. Во-вторых, выбирается мера расстояний $||x_i - c_j||$. Она характеризует степень близости точки к центру кластера. Если вектор характеристик состоит только из одного значения и задача, по сути, сводится к выделению значимых интервалов на прямой, то мера расстояния может иметь вид

$$||x_i - c_j|| = |x_i - c_j|.$$

Эта мера характеризует удаленность точки от центра кластера на прямой.

Третий параметр, который выбирается при инициализации – параметр сходимости алгоритма ε (уровень точности). Когда разность значений целевых функций текущей и предыдущей итераций достиг-

нет этого уровня, считается, что кластеризация завершена. Обычно этот параметр равен 0.001.

Четвертый параметр задается во избежание зависания алгоритма при невозможности достижения уровня точности. Это количество итераций алгоритма. Например, 10 000.

После выбора параметров генерируется случайным образом первоначальная матрица принадлежности объектов кластерам и происходит переход ко второму шагу алгоритма.

На шаге 2 происходит вычисление центров кластеров следующим образом:

$$c_j = \frac{\sum_{i=1}^{N} u_{ij}^{1.6} \cdot x_i}{\sum_{i=1}^{N} u_{ij}^{1.6}},$$

где c_j – центр j-го кластера; N – количество объектов; x_i – значение i-го объекта; u_{ij} – степень принадлежности объекта i кластеру j.

На третьем шаге формируется новая матрица принадлежности с учетом вычисленных на предыдущем шаге центров кластеров:

$$u_{ij} = \frac{1}{\sum_{l=1}^{c} \left(\frac{\|x_i - c_j\|}{\|x_i - c_l\|}\right)^{3.33}},$$

где u_{ij} – степень принадлежности объекта i кластеру j; c – количество кластеров; c_j – вектор центра j-го кластера; c_l – вектор центра l-го кластера.

При этом если для некоторого кластера j и некоторого объекта i расстояние $|x_i - c_j|=0$, тогда полагаем, что степень принадлежности u_{ij} равна 1, а для всех остальных кластеров степень принадлежности этого объекта равна нулю.

Далее на четвертом шаге вычисляется значение целевой функции, и полученное значение сравнивается со значением на преды-

дущей итерации. Если разность не превышает заданного в параметрах кластеризации значения ε, считаем, что кластеризация завершена. В противном случае переходим ко второму шагу алгоритма.

Мы рассмотрели методы нечеткой логики, теперь перейдем к следующей модели – искусственных нейронных сетей (ИНС).

Искусственные нейронные сети

Для подготовки данного раздела использовался материал (в том числе иллюстративный) из следующего источника: [26].

Особенности нейронных сетей

Прежде чем перейти к описанию того, что собой представляет модель искусственной нейронной сети, акцентируем внимание на особенностях данного подхода, а в частности его достоинствах и недостатках. Это основные моменты, которые обязательно необходимо понять и запомнить.

Достоинства ИНС:

1. Нелинейность модели, что дает возможность аппроксимировать любые нелинейные функции. Та же задача с помощью полиномиальной модели не всегда решается, либо же ее решение становится низким по качеству.

2. Локальности восприятия, заключающаяся в том, что каждый нейрон получает не весь входной вектор. Это позволяет ей сегментировать данные и работать с более сложными ситуациями.

3. Каскад слоев. В сочетании с пунктом 2 это дает способность воспринимать более абстрактные признаки.

4. Лучше работает с мультиколлинеарностью и комбинациями признаков.

5. Нейронная сеть дает несколько механизмов для контроля переобучения.

6. Нейронная сеть способна дообучаться при непротиворечивости новых образов.

Недостатки:

Нейронная сеть не интерпретируема. Поэтому не ясна логика преобразования входных данных в выходные. Не всегда можно убедиться в стабильности решения на всей области определения.

Математика процессов и некоторых аспектов функционирования еще недостаточно изучена. Часто количество нейронов и слоев приходиться подбирать экспериментально для конкретной задачи, потому что определенной методики нет.

Нет аналитического решения, а решение численными методами градиентного спуска может приводить к попаданию в локальные минимумы ошибки.

Для ИНС очень часто требуется большая выборка. Так для распознавания речи одного языка может потребоваться от 5 до 10 тыс. часов размеченных записей. Для Глубоких сетей необходимо еще большее число объектов обучающей выборки, иначе будет постоянно происходить переобучение.

Обучение большой ИНС требует больших вычислительных ресурсов и специального оборудования GPU.

Поскольку на сегодняшний день нейронные сети очень модная тема, следует отметить, что ИНС, так же как любые обобщающие модели машинного обучения, чувствительна к противоречиям в данных и, конечно же, на текущем этапе развития НС не могут содержать противоречивые «представления» об объекте, проводить какие-то внутренние размышления и перестановки. Иными словами, из неоткуда информацию ИНС не возьмет (самостоятельно не проделает ряд выводов).

Так одним из главных конкурентов ИНС в ряде задач являются решающие деревья и ансамбли над решающими деревьями (например, XGBoost). Исключением являются задачи машинного зрения и работы с изображениями вообще, в которых ИНС занимаются сегодня лидирующие позиции.

Определение модели искусственной нейронной сети

Согласно Википедии [1]: «Искусственная нейронная сеть – математическая модель, а также ее программное или аппаратное воплощение, построенная по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма. Это понятие возникло при изучении процессов, протекающих в мозге, и при попытке смоделировать эти процессы. Первой такой попыткой были нейронные сети, созданные У. Мак-Каллоком и У. Питтсом. После разработки алгоритмов обуче-ния получаемые модели стали использовать в практических целях: в задачах прогнозирования, для распознавания образов, в задачах управления и др.».

Искусственные нейроны сильно упрощены по сравнению с их биологическими прототипами. Каждый из них имеет дело только с сигналами, которые он периодически получает, и сигналами, которые он периодически посылает другим нейронам. Способность решать довольно сложные задачи обуславливается тем, что все нейроны соединены в достаточно большую сеть с управляемым взаимодействием. Схему простой нейронной сети можно увидеть на рисунке 24. Литерой «I» обозначены входные нейроны, «S» – скрытые нейроны, «O» – выходной нейрон. Соединяющие линии – связи между нейронами, или синаптические связи. Именно синаптические связи играют важнейшую роль в модели ИНС, так как в основе обучения лежит механизм корректировок силы этих связей.

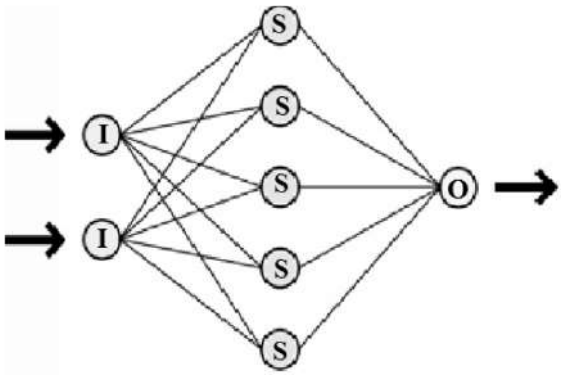


Рисунок 24. Схема простой нейросети

Нейронную сеть можно рассмотреть с разных точек зрения. Например, в машинном обучении, нейронная сеть – частный случай методов распознавания образов, дискриминантного анализа, методов кластеризации и т. п. С точки зрения математики, обученная нейронная сеть – решенная многопараметрическая задача нелинейной оптимизации. Кибернетика представляет нейронную сеть как «черный ящик» или произвольную передаточную функцию, которую можно получить при обучении сети и затем воспроизводить при использовании сети. С точки же зрения искусственного интеллекта, ИНС является основой философского течения коннективизма и основным направлением в структурном подходе по изучению возможности построения (моделирования) естественного интеллекта с помощью компьютерных алгоритмов.

Очень часто именно про нейронные сети говорится, что они не программируются в привычном смысле этого слова, они обучаются и что возможность обучения – одно из главных преимуществ нейронных сетей перед традиционными алгоритмами. Конечно же, такие трактовки и объяснения нельзя назвать точными и их можно допускать только при первом знакомстве с машинным обучением вообще.

Термин «обучение» необходимо трактовать лишь математически. В таком случае и Регрессия, и Деревья также обучаются, хотя в любой модели происходит одно и то же – корректировка весов.

Однако шумиха вокруг нейронных сетей порождает массу мифов, принижающих остальные методы, изученные нами ранее.

В чем-то эта модель напоминает уже изученные модели, такие как Регрессия и Деревья решений. В каком-то смысле Нейронные сети являются комбинацией этих методов (на уровне концепции).

В основе математической модели вычислений в ИНС лежит взвешенное суммирование, то есть, по сути, уже знакомое вам полиномиальное уравнение (в котором коэффициенты уже принято называть весами). Веса этого полинома и отражают силу синаптической связи между нейронами. Именно синаптические связи играют важнейшую роль в модели ИНС, так как в основе обучения лежит

механизм корректировок силы этих связей (подобно биологическому прототипу).

Все также вычисления завязаны на понятии ошибки и на том принципе, что корректное обобщение (отображение X=>Y) строится посредством уменьшения этой ошибки. Таким образом, они являются логическим продолжением прошлых тем.

Однако стоит отметить, что в отличие от методов Байеса и Регрессии нейронные сети изначально разрабатывались исходя из кибернетического подхода к моделям и информации. То есть нейронные модели – это некая адаптивная система, которая может прийти из одного состояния в требуемое, но при этом имеется некая случайная составляющая, которая вносит нестабильность в решения. Также нет хорошей компактной формулы, описывающей работу сети, – с самого начала суть вычислений в нейронных сетях была основана на численном (постепенном\итеративном) подходе к уменьшению ошибки.

Первая формальная модель и первая реализация нейронной сети

Согласно [27]: «Первой формальной моделью нейронных сетей (НС) и нейрона вообще была модель МакКаллока-Питтса, уточненная и развитая Клини. Впервые было установлено, что НС могут выполнять любые логические операции и вообще любые преобразования, реализуемые дискретными устройствами с конечной памятью. Эта модель легла в основу теории логических сетей и конечных автоматов и активно использовалась психологами и нейрофизиологами при моделировании некоторых локальных процессов нервной деятельности. В силу своей дискретности она вполне согласуется с компьютерной парадигмой и, более того, служит ее «нейронным фундаментом».

Пусть имеется n входных величин $x_1,\ldots,x_n$ бинарных признаков, описывающих объект x. Значения этих признаков будем трактовать как величины импульсов, поступающих на вход нейрона через n входных синапсов. Будем считать, что, попадая в нейрон, импульсы складываются с весами $\omega_1,\ldots,\omega_n$.

Если вес положительный, то соответствующий синапс возбуждающий, если отрицательный, то тормозящий. Если суммарный импульс превышает заданный порог активации ω_0, то нейрон возбуждается и выдает на выходе 1, иначе выдается 0.

Таким образом, нейрон вычисляет n-арную булеву функцию

$$a(x) = \varphi(\sum_{j=1}^{n} w_j x^j - w_0),$$

где $\varphi(z) = [z \geq 0]$ – ступенчатая функция Хевисайда.

В теории нейронных сетей функцию φ, преобразующую значение суммарного импульса в выходное значение нейрона, принято называть функцией активации. Таким образом, модель МакКаллока-Питтса эквивалентна пороговому линейному классификатору. Схема ее вычислений показана на рисунке 25.

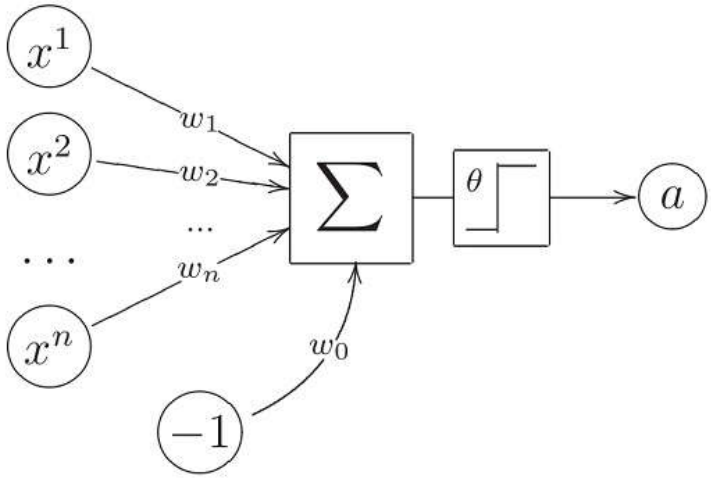


Рисунок 25. Схема вычислений в модели нейрона МакКаллока-Питтса

Получается, что нейрон производит взвешивание входных признаков (то есть рассчитывает результат с учетом их важности\веса) подобно регрессии. Однако важно понимать, что это пока всего лишь один нейрон.

Теоретические основы нейроматематики были заложены в начале 40-х годов, и в 1943 году У. МакКаллок и его ученик У. Питтс сформулировали основные положения теории деятельности головного мозга.

Ими были получены следующие результаты:

- разработана модель нейрона как простейшего процессорного элемента, выполняющего вычисление переходной функции от скалярного произведения вектора входных сигналов и вектора весовых коэффициентов;
- предложена конструкция сети таких элементов для выполнения логических и арифметических операций;
- сделано основополагающее предположение о том, что такая сеть способна обучаться, распознавать образы, обобщать полученную информацию.

Недостатком данной модели является то, что в качестве функции активации используется функция Хевисайда или униполярная пороговая функция (рисунок 26). В формализме, предложенном МакКаллоком и Питтсом, нейроны имеют состояния 0, 1 и пороговую логику перехода из состояния в состояние. Каждый нейрон в сети определяет взвешенную сумму состояний всех других нейронов и сравнивает ее с порогом, чтобы определить свое собственное состояние.

$$f(z)=\begin{cases}1 & при \quad z\geq\alpha;\\ 0 & при \quad z<\alpha.\end{cases}$$

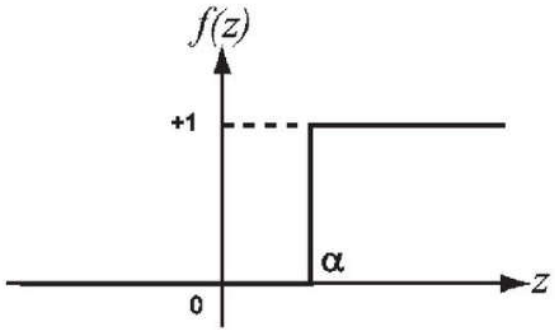


Рисунок 26. Пороговая функция, или функция Хевисайда

Пороговый вид функции не предоставляет нейронной сети достаточную гибкость при обучении и настройке на заданную задачу. Если значение вычисленного скалярного произведения, даже незначительно, не достигает до заданного порога, то выходной сигнал не

формируется вовсе, и нейрон «не срабатывает». Это значит, что теряется интенсивность выходного сигнала (аксона) данного нейрона и, следовательно, формируется невысокое значение уровня на взвешенных входах в следующем слое нейронов.

К тому же модель не учитывает многих особенностей работы реальных нейронов (импульсного характера активности, нелинейности суммирования входной информации, рефрактерности).

Несмотря на то, что за прошедшие годы нейроматематика ушла далеко вперед, многие утверждения МакКаллока остаются актуальными и поныне. В частности, при большом разнообразии моделей нейронов принцип их действия, заложенный МакКаллоком и Питтсом, остается неизменным».

Первой реализацией модели нейронной сети была созданная в 1960 электронная машина «Марк-1». Идею предложил Фрэнк Розенблатт в 1957. Согласно Википедии [1]: «Несмотря на то, что это первая модель и концепция, тем ни менее многие идеи современных нейронных сетей во многом совпадают с концепцией персептрона. Так выше была рассмотрена модель упрощенного искусственного нейрона, способного производить вычисления. Персептрон по своей сути является сетью таких нейронов. Из чего и предполагалось получать механизм для более сложных вычислений.

Несмотря на свою простоту, персептрон способен обучаться и решать довольно сложные задачи. Основная математическая задача, с которой он справляется, – это линейное разделение любых нелинейных множеств, так называемое обеспечение линейной сепарабельности. Логическая схема персептрона с тремя выходами представлена на рисунке 27.

Персептрон состоит из трех типов элементов, а именно: поступающие от датчиков сигналы передаются ассоциативным элементам, а затем реагирующим элементам. Таким образом, персептроны позволяют создать набор «ассоциаций» между входными стимулами и необходимой реакцией на выходе.

В биологическом плане это соответствует преобразованию, например, зрительной информации в физиологический ответ от двигательных нейронов. Согласно современной терминологии, персептроны могут быть классифицированы как искусственные нейронные сети:

- с одним скрытым слоем;
- с пороговой передаточной функцией;
- с прямым распространением сигнала».

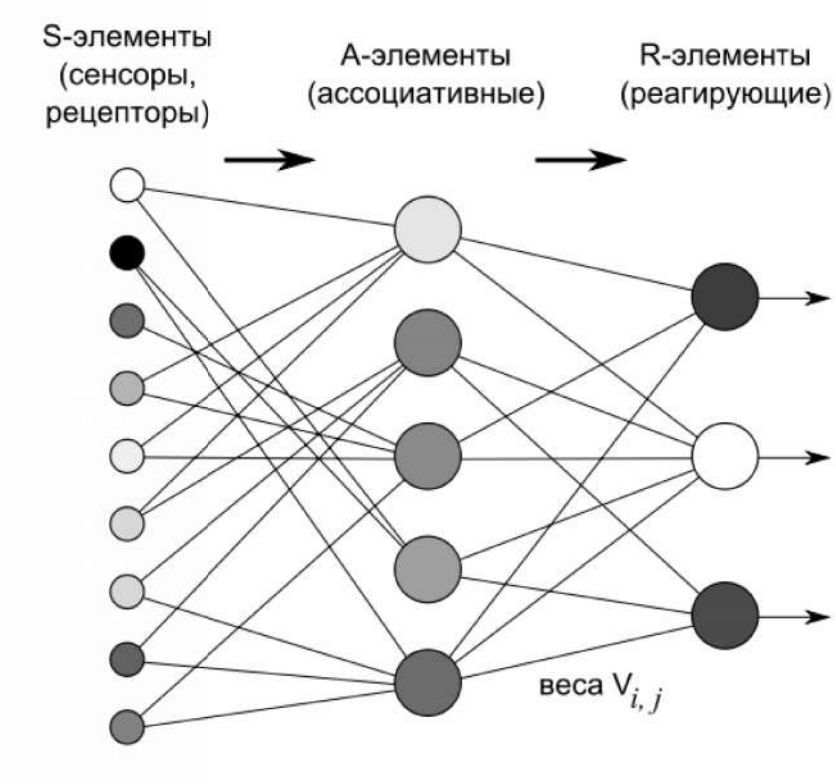


Рисунок 27. Логическая схема персептрона с тремя выходами

Многослойный персептрон (MLP)

Данный тип\архитектура является самой распространенной архитектурой искусственных нейронных сетей и в каком-то смысле классической. Очень часто, когда речь идет о нейронных сетях вообще, то имеется в виду именно многослойная сеть прямого распространения. Многие задачи классификации, аппроксимации и управления, которые решаются с помощью нейронных сетей, реша-

ются именно этим типом сети. Кроме того, принципы обучения, разработанные для этого типа сети, впоследствии стали применяться и для других типов. Так что в каком-то смысле это базовая архитектура для других типов сетей.

Если подходить к определению строго, то не совсем корректно называть такой тип сети Персептроном, ведь она отличается от него не только количеством слоев, но и тем, что:

- Нейроны имеют не ступенчатую функцию активации Хевисайда, а сигмоидальную функцию.
- Обучение производится не по правилу Хебба, а с помощью обратного распространения ошибки.

Поэтому такую архитектуру называют либо просто многослойной сетью, либо сетью прямого распространения или многослойной сетью прямого распространения. Но в то же время можно встретить и более удобное название MLP, что значит многослойный персептрон. Такое название определяет, что это все-таки не просто сеть. Структура MLP представлена на рисунке 28.

Замечание: выше уже упоминалось о том, что терминология в машинном обучении и в нейросетевых технологиях в частности еще не до конца систематизирована. Поэтому необходимо разбираться в сути моделей, допуская определенную вариативность названий. Однако со временем это перестает доставлять какие-либо трудности.

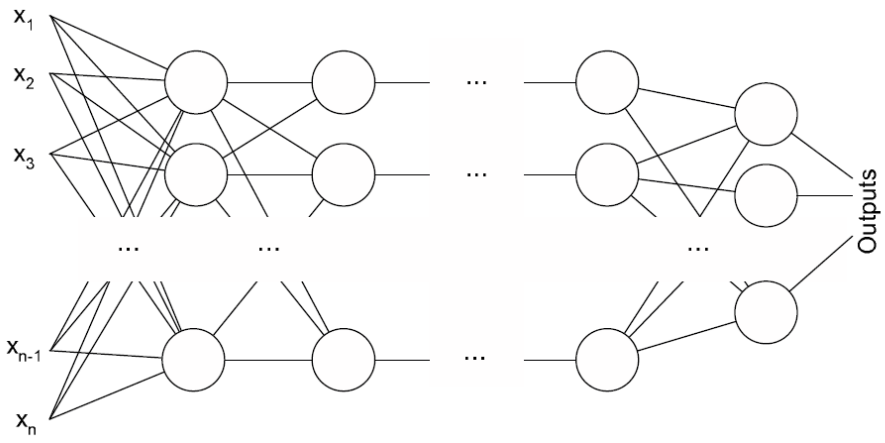


Рисунок 28. Структура многослойного персептрона

На рисунке 29 представлен график сигмоидальной функции активации.

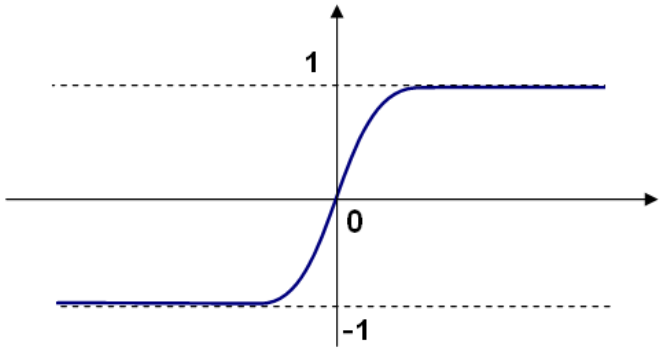


Рисунок 29. График сигмоидальной функции

Сигмоидальная функция активации обладает следующими преимуществами по сравнению с пороговой функцией Хевисайда:

Нелинейность. Обеспечивает модели возможность обрабатывать (воспринимать) нелинейные закономерности в данных.

Фиксированные ограничения выхода. Это особенность позволяет масштабировать данные от слоя к слою.

Непрерывность и Дифференцируемость. Данные свойства позволяют обойти главный недостаток пороговой функции – резкий переход. Поскольку обучение сети происходит посредством градиентного спуска, то движение по пороговой функции слишком резкое, в отличие от сигмоидальной функции – движение по градиенту которой плавное и, как следствие, нахождение минимума ошибки становится более стабильным и вероятным.

Замечание 1: Кроме того, что нелинейная функция активации позволяет модели обрабатывать нелинейные закономерности в данных, она выполняет и другую важную задачу. Нелинейные выходы одного слоя нейронов позволяют подключить к нему второй, третий и т. п. слои. Если сделать много слоев в Персептроне с линейной функцией активации или с пороговой (суть в том, что она не непрерывна), то можно свести все эти слои всего лишь к одному слою

(с математической точки зрения последовательность взвешенных сумм (полиномов) можно легко свести к одному другому полиному).

Замечание 2: Подобно регрессионным моделям обучение модели ИНС подразумевает правку только весовых коэффициентов. Никакие другие процессы или изменения в модели не происходят.

Алгоритм обратного распространения ошибки как специальный метод градиентного спуска, разработанный именно для ИНС, будет подробнее разобран далее.

Сверточные (ConvolutionalNeuralNet) и Глубокие (DeepNet) Сети

Одна из базовых способностей людей и животных заключается в том, что мы легко распознаем визуальные образы под любым углом и в любом положении. Буква «А» останется для нас буквой «А» в какой бы области видимости наших глаз мы ее не увидели. Конечно же, при условии того, что выдержан определенный порог зашумленности, освещенности и т. п. Даже если текст расположен вверх ногами, то довольно быстро можно приноровиться его читать. А уж небольшие наклоны или разные шрифты на рекламных щитах вообще не представляют для нас проблемы.

А вот системы по распознаванию текста, к сожалению, не обладают такими возможностями. Либо они заточены распознавать определенный тип шрифта, либо они очень чувствительны к сдвигам или наклонам, либо к масштабу, либо вообще ко всему. И несмотря на то, что нейронные сети в качестве прототипа использовали принципы работы мозга, долгое время не было хорошего решения этой проблемы. Данный недостаток называется проблемой чувствительности к пространственным искажениям. А добиться нужно так называемой инвариативности к таким искажениям трех основных типов: смещениям, поворотам, масштабированию.

По поводу пространственных искажения и появления сверточной сети, в источнике [28] сказано следующее:

«Дело в том, что работа зрительной коры гораздо сложнее: в ней происходит анализ и цвета, и текстуры, и движения; информация от

двух глаз объединяется для осуществления стереозрения; работает множество других механизмов. Высшие зрительные функции все еще остаются загадкой. И самое главное, до сих пор не известно, как происходит обучение. До конца неясно даже, что в структуре зрительной системы заложено генетически, а что формируется под влиянием опыта. Хотя структура зрительной системы у человека продолжает формироваться до 4–5 лет, это может быть, как реализацией генетической программы, лишь немного адаптирующейся к окружению, так и детальным обучением, в результате которого создаются основные связи зрительного тракта.

Пытаясь разработать ИНС, которая была бы еще ближе к биологическому прототипу, японский ученый-компьютерщик Кунихика Фукусима предлагает принципиально новую модель Когнитрона в 1975 г., а в 1980 г. модификацию этой модели – «Неокогнитрон». Главное отличие этих сетей от MLP заключалось в том, что слои нейронов упаковывались не в линию, а в двумерную плоскость, чтобы информация циркулировала не только от слоя к слою, но еще сохраняла определенную пространственную ориентацию (рисунок 30). Кроме того, в этих сетях использовались принципы самоорганизации, то есть обучение происходило без учителя».

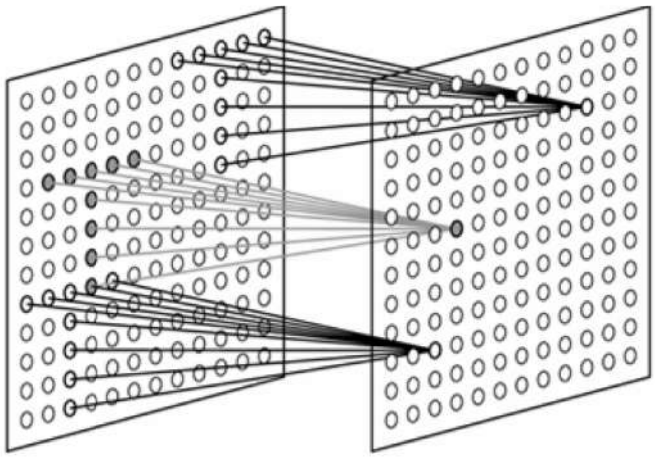


Рисунок 30. Рецептивные поля (квадратные плоскости) простых клеток, настроенных на поиск выбранного паттерна в разных позициях

В итоге подобная архитектура дала хорошие результаты при распознавании символов и даже рукописного текста относительно классических нейронных сетей. Однако по сравнению другими специализированными алгоритмами (не машинного обучения) на тот момент неокогнитрон «не дотягивал». Требовалось усложнять модель, но тогда были явные недостатки по скорости обучения и работы.

Однако общая концепция пространственно-ориентированных карт была очень ценной. И французский ученый и специалист в области обработки сигналов и компьютерного зрения Ян ЛеКун предлагает более упрощенную модель сети. Он убрал из неокогнитрона функции, которые нужны были только для того, чтобы быть похожим на реальный мозг. Он также показал, как можно использовать метод обратного распространения ошибки для обучения сетей, архитектура которых, как и у неокогнитрона, отдаленно напоминает строение коры мозга. И этот способ обучения, уже хорошо проверенный на тот момент, оказался куда эффективнее самоорганизации сетей. Так в 1995 появились нейронные сети Сверточного типа.

Структура сверточных сетей представлена на рисунке 31.

Нейроны здесь также упакованы не только в слои, но и двумерные карты. Информация циркулирует слева на право по нейронам такого же типа (как и в MLP). Но главная особенность заключается в том, что сеть не полносвязная, то есть каждый нейрон имеет свою небольшую область видимости (на верхнем рисунке область видимости показана пунктирными линиями). Такое локальное восприятие и обобщение от слоя к слою и дает решение проблемы чувствительности к пространственным искажениям, о которых говорилось выше. Иными словами, Сверточная Нейронная Сеть (СНС) способна обрабатывать пространственную топологию.

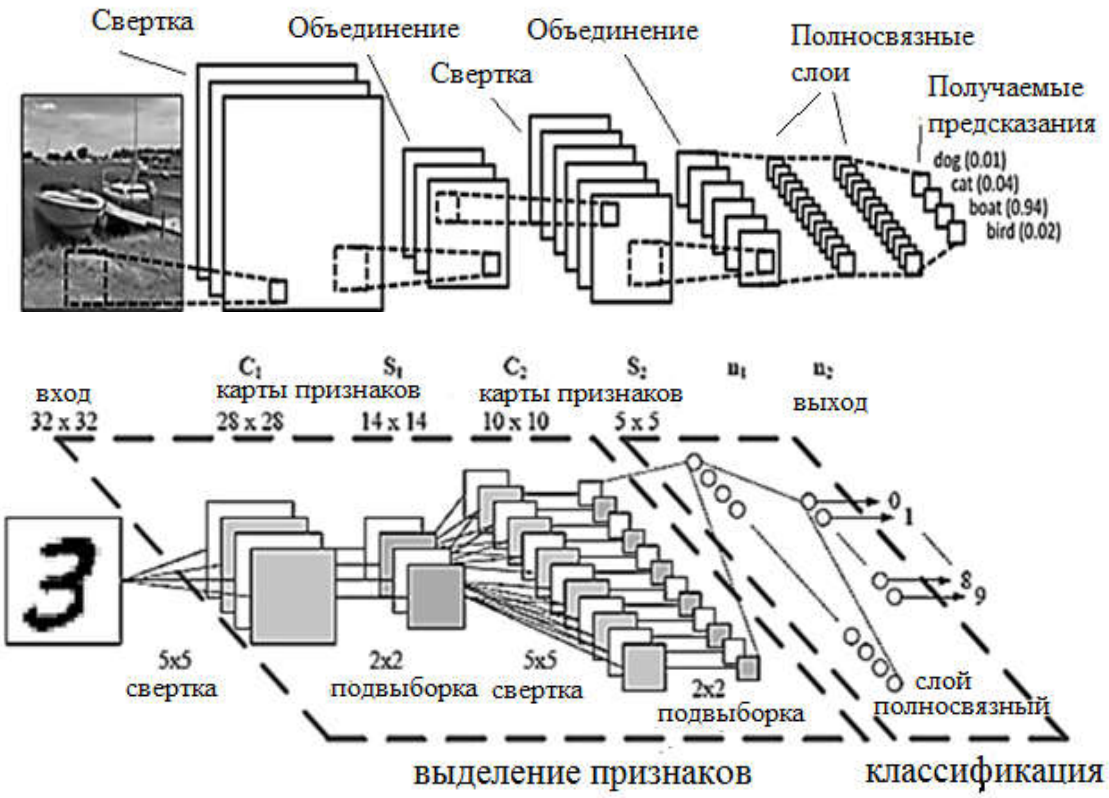


Рисунок 31. Структурная схема Сверточных нейронных сетей

Именно эта архитектура ИНС легла в основу так называемых Глубоких сетей, что породило понятие глубокого обучения (Deep Learning). Именно сеть этой архитектуры произвела такую шумиху в 2007 г., приблизив качество распознавания текста к человеческому уровню. Безусловно, данный метод не лишен недостатков, но во многих задачах именно визуального распознавания СНС является лучшим решением.

Глубокие сети, по сути, являются уже четвертой вехой развития когнитрона и их главная особенность заключается в многослойности (рисунок 32).

Но кроме этого важную роль играет разнообразное комбинирование определенных блоков карт. В глубоких сетях очень много разных модулей и ответвлений (может быть и такое, что сеть имеет несколько входов на разных уровнях).

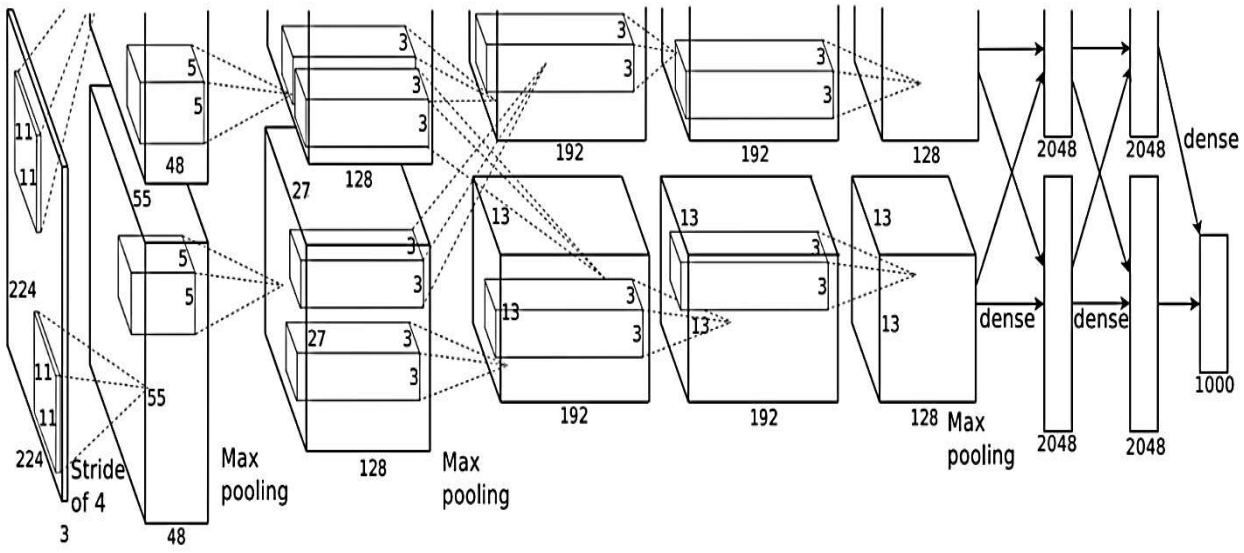


Рисунок 32. Схематичное представление Глубокой сети

Однако наращивание слоев приводит к целому ряду трудностей, как с объемом вычислений, так и с обучением сети, из чего складываются особенности, присущие именно глубокому обучению.

Карты (ART, SFAM)

Данный тип нейронных сетей разрабатывался как решение проблемы «пластичности-стабильности». Как известно, взрослый человек может понять какую-либо информацию с одного или нескольких объяснений. Речь, конечно, не идет о сложных вещах вроде курса математического анализа, но, тем не менее, мозг человека способен понять какие-то концепции даже за одно объяснение. Если переводить это на язык машинного обучения, то это означает, что нам не нужно многократное предъявление схожих примеров, нам хватит одного-двух объектов обучающей выборки. Конечно, мозг способен на такие вещи не только или, правильнее будет сказать, не столько благодаря быстро обучающимся нейронам, а потому что, воспринимая чье-то объяснение (например, когда вы читаете инструкцию к телевизору) мозг уже имеет представление о многих вещах. Иными словами, мозг уже обучен, у него есть предыдущий опыт. Кроме того,

разум способен держать контекст и т. п. Если объяснять упрощенно, то мозг дообучается, а не переобучается в таких случаях.

Но даже если не принимать во внимание контекст и априорный опыт, то возможно ли сделать обучение сети столь же пластичным, ну или близким к этому? Первая задача разработчиков сетей ART заключалась в том, чтобы сделать процесс обучения быстрым, чтобы не проходить итеративный градиентный спуск для правки весов. Эта проблема и есть проблема пластичности ИНС.

Но была и вторая задача. Опять же, вдохновляясь биологическим прототипом, разработчики понимали, что если человек узнает, что по какому-то вопросу у него на протяжении некоторого времени была неверная точка зрения, то он достаточно просто скорректирует отдельные представления о мире. У него не разрушится вся картина мира, он не сойдет с ума и не потеряет другие знания. Конечно, тут мы не берем в расчет психологические факторы, которые помешают ему это сделать.

Но если не рассматривать такие ситуации, то можно принять тот факт, что мозг очень пластичен, то есть может обучится\дообучиться с малого количества раз и в то же время стабилен – одни знания могут быть заменены на другие без разрушения другой информации.

В то же время информация в классических ИНС (в MLP) распределяется по весам всей сети. Иными словами, нет одного четкого места, которое бы отвечало за определенный блок информации. Таким образом, если дообучение происходит на объектах тех же классов, которые использовались во время обучения, то многослойную сеть прямого распространения можно дообучить (но опять же медленно). Но если взять обученную сеть на одних классах и попытаться до обучить на новых, то вся старая информация начнет разрушаться.

Например, ИНС типа MLP была обучена для классификации квадратов и кругов по визуальным образам. Допустим, что сеть стала показывать хорошие показатели качества после предъявления различных образов квадратов и кругов в количестве 10 тыс. штук. Если

позже дообучить сеть на новых вариантах квадратов и кругов, то сеть может еще улучшить показатели. Но если начать обучать ее еще и на треугольниках, то под этот новый класс не выделится определенное место в памяти сети, а распределенная по весам информация, наоборот, начнет меняться, разрушая обученное состояние.

Это и есть «проблема стабильности ИНС». Она была второй задачей разработчиков сетей типа ARTMAP (Творческие карты, или просто Карты).

Первые варианты Карт (ARTMAP 1, ARTMAP 2) были очень сложными, избыточными и во многом были инженерным творением в виде электрических схем, а не алгоритмом.

После возвращения интереса к нейронным сетям этот тип сетей был существенно оптимизирован и модифицирован. Один из самых успешных и эффективных вариантов называется Simplified FUZZY ARTMAP (упрощенная нечеткая Карта). Под нечеткостью здесь понимается то, что вычисления основаны на нечеткой логике. Детали работы нечеткости в этом подразделе не приводятся.

Структура такой сети представлена на рисунке 33.

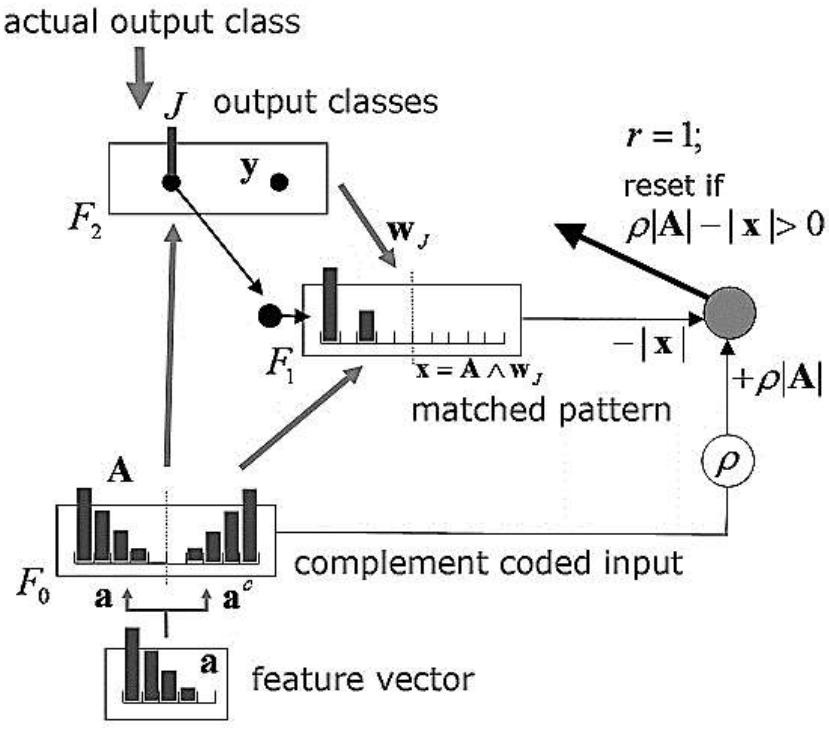


Рисунок 33. Структура основных модулей системы ARTMAP

Собственно, правильнее будет назвать это системой, содержащей нейроны, так как в структуре есть множество сторонних блоков (узлов), которые не очень красиво вписываются в биологическую концепцию мозга (хотя и не противоречат ей).

Итак, рассмотрим представленную на рисунке структуру снизу-вверх. В самом низу имеем вектор признаков «a» (как и во всех других методах ML). Данный вектор обязательно должен быть нормализован (отмасштабирован), то есть сеть не просто очень чувствительна к ненормализованным данным, а вообще не работает без нормализации. Соответственно если природа каких-то признаков неизвестна и не ясно, какой диапазон может быть в данных, то применение этой сети затрудняется (нужно специальным образом контролировать данные на вход) или вовсе отменяется.

Далее нормализованный вектор *a* дополняется комплементарной (обратной) парой a^c: там, где в исходном векторе было значение 0, в обратной паре будет 1, и наоборот (при условии нормализации от 0 до 1). Для этого и была необходима нормализация.

Далее, полученный вектор сравнивается со всеми шаблонами в базе нейронной сети. В данной случае это именно база, а не просто распределенная информация. Один нейрон в такой архитектуре представляет собой одну единицу информации, один конкретный шаблон\образ. То есть один нейрон выражает какой-то образ обучающей выборки (например, квадрат или круг). При обучении каждый поступающий вектор сравнивается с уже имеющейся базой образов, которые сохранены в нейронах сети.

Если близких образов нет, то образ сохраняется как новый, то есть создается новый нейрон и вектор входного образа полностью копируется в веса нового нейрона. Так с одного раза происходит полное корректное запоминание. При этом за нейроном закрепляется метка y (то есть это по-прежнему обучение с учителем, и для каждого входного вектора имеется ответ).

Если в базе уже имеется образ, похожий на входной образ, то происходит следующее. Образ в базе, выраженный нейроном и его

весами, подтягивается\приближается ко входному. Допустим, у сети в базе уже имеется некое представление о квадрате, и мы подаем еще один похожий образ квадрата, тогда после подтягивания в базе будет храниться нечто среднее, какой-то усредненный образ этих двух квадратов. Соответственно надо четко понимать, по какими принципам рассчитывается это усреднение. Ведь это все-таки не искусственный интеллект, и по одному образу человека и дельфина, сеть не усреднит их в образ млекопитающих. Надо понимать, что это лишь векторное усреднение. То есть если входной образ (0.3, 0.7), а образ в базе (0.35, 0.8), то итоговый будет (0.325, 0.75). Что с точки зрения гипотезы компактности позволяет решать задачу обобщения.

Замечание: напомним, что гипотеза компактности говорит нам о том, что объекты одного класса обладают близкими по значению признаками (или какой-либо комбинацией признаков), из-за чего их вообще можно отделить от объектов другого класса. Если пойти еще дальше, то можно сказать, что объекты сходных классов располагаются в некотором N-мерном пространстве признаков также близко. Тут можно сказать, что объекты потому и принадлежат какому-то одному смысловому классу, потому что по какому-то признаку или группе (комбинации) похожи.

Итак, данная сеть способна сохранять образ с одного предъявления, что решает проблему пластичности. В сети сразу появляется образ с меткой класса, к которому принадлежит этот образ. Таким образом, вместо минимизации ошибки данная сеть обучается путем создания похожих прототипов и сравнением входного образа с прототипами. В рабочем режиме сеть выдает ответ Y в виде метки того нейрона, образ которого сильнее всего похож на текущий входной образ.

При этом данная сеть также решает задачу стабильности, так как теперь есть четкие хранилища информации и система организации новых нейронов. Так что даже при предъявлении новых классов сеть будет стабильно работать.

Данный тип сети хорошо зарекомендовал себя в медицине и при распознавании звуковых сигналов. Однако есть и ряд минусов данной сети:

- при равной степени обобщения сеть ARTMAP потребует существенно большее количество памяти и вычислений, чем MLP, если речь идет о входных векторах большой размерности и очень больших обучающих выборках;
- сеть не способна учитывать топологию пространства и сложные абстрактные признаки (так как слоев по сути нет).

Рекуррентные сети (Recurrent Neural Network)

Рекуррентные нейронные сети (RecurrentNeuralNetwork; RNN) – вид нейронных сетей, в которых имеется обратная связь. При этом под обратной связью подразумевается связь от логически более удаленного элемента к менее удаленному (рисунок 34). Наличие обратных связей позволяет запоминать и воспроизводить целые последовательности реакций на один стимул.

В сетях такого типа возникает эффект памяти и способности воспринимать не только статичный образ, но и динамику образов (так как есть возможность учитывать историю через обратную связь).

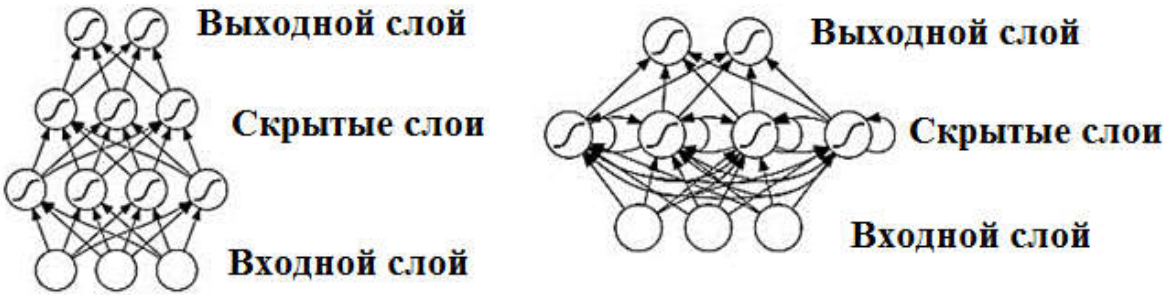


Рисунок 34. Сеть прямого распространения (слева) и рекуррентная сеть (справа)

Согласно Википедии [1]: «Долгая краткосрочная память (англ. Longshort-termmemory; LSTM) – разновидность архитектуры рекуррентных нейронных сетей (RNN), предложенная в 1997 году Сеппом Хохрайтером и Юргеном Шмидхубером. Как и большинство рекур-

рентных нейронных сетей, LSTM-сеть является универсальной в том смысле, что при достаточном числе элементов сети она может выполнить любое вычисление, на которое способен обычный компьютер, для чего необходима соответствующая матрица весов, которая может рассматриваться как программа. В отличие от традиционных рекуррентных нейронных сетей, LSTM-сеть хорошо приспособлена к обучению на задачах классификации, обработки и прогнозирования временных рядов в случаях, когда важные события разделены временными лагами с неопределенной продолжительностью и границами.

Относительная невосприимчивость к длительности временных разрывов дает LSTM преимущество по отношению к альтернативным рекуррентным нейронным сетям, скрытым Марковским моделям и другим методам обучения для последовательностей в различных сферах применения. Из множества достижений LSTM-сетей можно выделить наилучшие результаты в распознавании несегментированного слитного рукописного текста и победу в 2009 году на соревнованиях по распознаванию рукописного текста (ICDAR). LSTM-сети также используются в задачах распознавания речи, например, LSTM-сеть была основным компонентом сети, которая в 2013 году достигла рекордного порога ошибки в 17,7% в задаче распознавания фонем на классическом корпусе естественной речи TIMIT. По состоянию на 2016 год, ведущие технологические компании, включая Google, Apple, Microsoft и Baidu, используют LSTM-сети в качестве фундаментального компонента новых продуктов».

Самоорганизующиеся карты (Self-organization map, SOM)

Согласно ресурсу [29]: «Такие сети представляют собой соревновательную нейронную сеть с обучением без учителя, выполняющую задачу визуализации и кластеризации. Является методом проецирования многомерного пространства в пространство с более низкой размерностью (чаще всего, двумерное), применяется также для решения задач моделирования, прогнозирования и др. Является одной из версий нейронных сетей Кохонена. Самоорганизующиеся

карты Кохонена служат, в первую очередь, для визуализации и первоначального («разведывательного») анализа данных.

Геометрическая суть алгоритма такая, что близкие объекты в многомерном пространстве (схожие животные по тысяче признаков) будут расположены близко и на двумерной карте, где объекты будут представлены точками. Собственно, обучение сети это и есть процесс укладки таких точек (итеративного оттягивания таких точек от начальных позиций, см. рисунок 35).

Сигнал в сеть Кохонена поступает сразу на все нейроны, веса соответствующих синапсов интерпретируются как координаты положения узла, и выходной сигнал формируется по принципу «победитель забирает все», то есть ненулевой выходной сигнал имеет нейрон, ближайший (в смысле весов синапсов) к подаваемому на вход объекту. В процессе обучения веса синапсов настраиваются таким образом, чтобы узлы решетки «располагались» в местах локальных сгущений данных, то есть описывали кластерную структуру облака данных, с другой стороны, связи между нейронами соответствуют отношениям соседства между соответствующими кластерами в пространстве признаков».

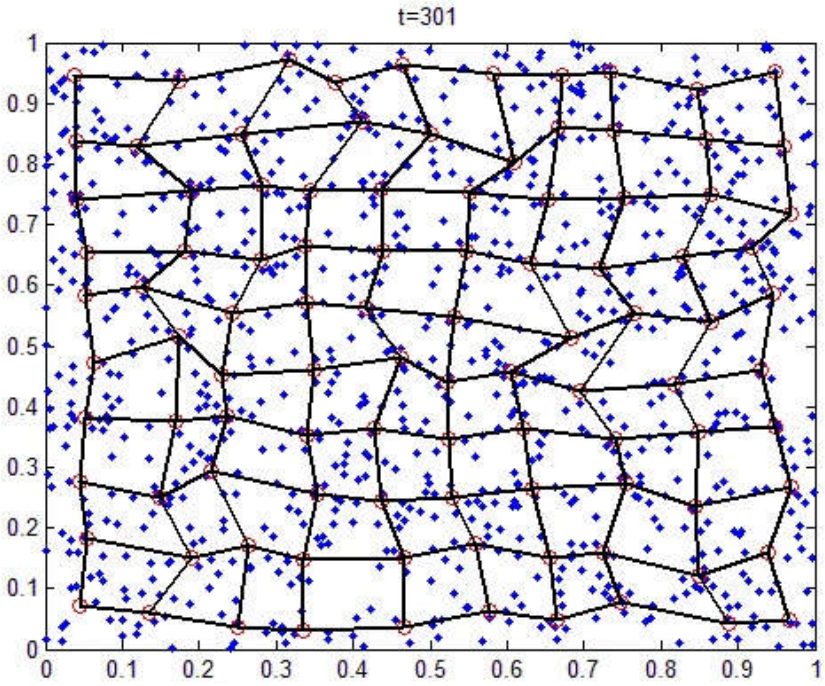


Рисунок 35. Визуализация двумерной плоскости с точками (проекциями) объектов в SOM

Автокодировщики (AutoEncoder)

Согласно Википедии [1]: «Автокодировщики (AutoEncoder) – специальная архитектура искусственных нейронных сетей, позволяющая применять обучение без учителя при использовании метода обратного распространения ошибки. Простейшая архитектура автокодировщика – сеть прямого распространения, без обратных связей, наиболее схожая с персептроном и содержащая входной слой, промежуточный слой и выходной слой. В отличие от персептрона, выходной слой автокодировщика должен содержать столько же нейронов, сколько и входной слой (см. рисунок 36).

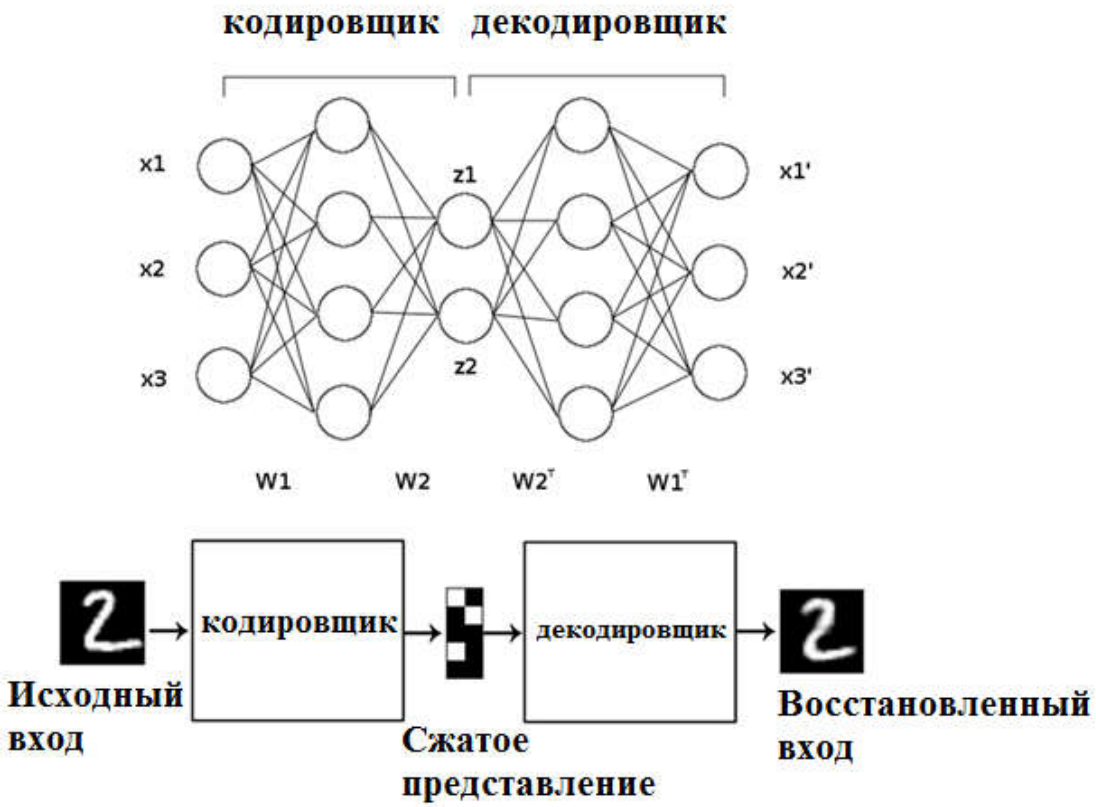


Рисунок 36. Схема сети типа Автокодировщик

Основной принцип работы и обучения сети автокодировщика – получить на выходном слое отклик, наиболее близкий к входному. Чтобы решение не оказалось тривиальным, на промежуточный слой автокодировщика накладывают ограничения: промежуточный слой должен быть или меньшей размерности, чем входной и выходной

слои, или искусственно ограничивается количество одновременно активных нейронов промежуточного слоя – разреженная активация. Эти ограничения заставляют нейросеть искать обобщения и корреляцию в поступающих на вход данных, выполняя при этом их сжатие. Таким образом, нейросеть автоматически обучается выделять из входных данных общие признаки, которые кодируются в значениях весов сети. Так, при обучении сети на наборе различных входных изображений, нейросеть может самостоятельно обучиться распознавать линии и полосы под различными углами.

Какое-то время автокодировщики применялись каскадно для обучения глубоких (многослойных) сетей, а именно для предварительного обучения глубокой сети без учителя. Для этого слои обучаются друг за другом, начиная с первых. К каждому новому необученному слою на время обучения подключается дополнительный выходной слой, дополняющий сеть до архитектуры автокодировщика, после чего на вход сети подается набор данных для обучения. Веса необученного слоя и дополнительного слоя автокодировщика обучаются при помощи метода обратного распространения ошибки. Затем слой автокодировщика отключается и создается новый, соответствующий следующему необученному слою сети. На вход сети снова подается тот же набор данных, обученные первые слои сети остаются без изменений и работают в качестве входных для очередного обучаемого автокодировщика слоя. Так обучение продолжается для всех слоев сети за исключением последних. Последние слои сети обычно обучаются без использования автокодировщика при помощи того же метода обратного распространения ошибки и на маркированных данных (обучение с учителем).

В последнее время автокодировщики мало используются для описанного «жадного» послойного предобучения глубоких нейронных сетей. После того как этот метод был предложен в 2006 г. Джеффри Хинтоном и Русланом Салахутдиновым, достаточно быстро оказалось, что новые методы инициализации случайными весами с определенным видом распределения оказываются эффективными

для улучшения сходимости. А предложенная в 2014 г. пакетная нормализация позволила обучать еще более глубокие сети, предложенный же в конце 2015 г. метод остаточного обучения позволил обучать сети произвольной глубины».

Основными практическими приложениями автокодировщиков остаются уменьшение шума в данных, а также уменьшение размерности многомерных данных для визуализации. Кроме того, с определенными оговорками, касающимися размерности и разреженности данных, автокодировщики могут позволять получать проекции многомерных данных, которые оказываются лучше тех, что дает метод главных компонент либо какой-либо другой классический метод. Помимо этого можно использовать такие сети, как алгоритм сжатия данных.

Крайне важно отметить, что если же взять архитектуру автокодировщика, но обучать его с учителем, то есть Y будет отличаться от X, то получится сеть для преобразования одного изображения в другое изображение через некоторую сложную функцию. На основе данного преобразования строятся различные фильтры изображений и видео (например, наложение различных эффектов) или же можно решать сложные задачи сегментации изображения. На сегодняшний день подобные архитектуры очень распространены и хорошо решают задачи анализа сцены для беспилотного автомобиля.

Импульсные (Спайковые) сети

Согласно Википедии [1]: «Первая научная модель импульсной нейронной сети была предложена Аланом Ходжкином и Эндрю Хаксли в 1952 году. Эта модель описывала, как потенциалы действия возникают и распространяются. Импульсы, однако, как правило, не передаются непосредственно между нейронами. Связь требует обмена химическими веществами, которые называются нейротрансмиттерами, в синаптической щели.

С точки зрения теории информации, проблема заключается в отсутствии модели, которая бы объясняла, как кодируется инфор-

мация и декодируются серии последовательностей импульсов, то есть потенциалы действия. Для нейробиологии все еще открытым является ответ на вопрос: нейроны связываются с помощью частотного или временного кодирования? С помощью временного кодирования один импульсный нейрон может заменять сотни скрытых элементов частотной нейронной сети.

Что же касается отличия спайковых сетей от классических, то тут необходимо отметить следующее. Если основное отличие сверточных сетей от классических заключается в структуре и организации связей, то импульсные сети, напротив, по структуре похожи на MLP. Их главное отличие заключается в том, что нейроны обмениваются короткими (у биологических нейронов – около 1-2 мс) импульсами одинаковой амплитуды (у биологических нейронов – около 100 мВ). Является самой реалистичной, с точки зрения физиологии, моделью ИНС (см. рисунок 37)».

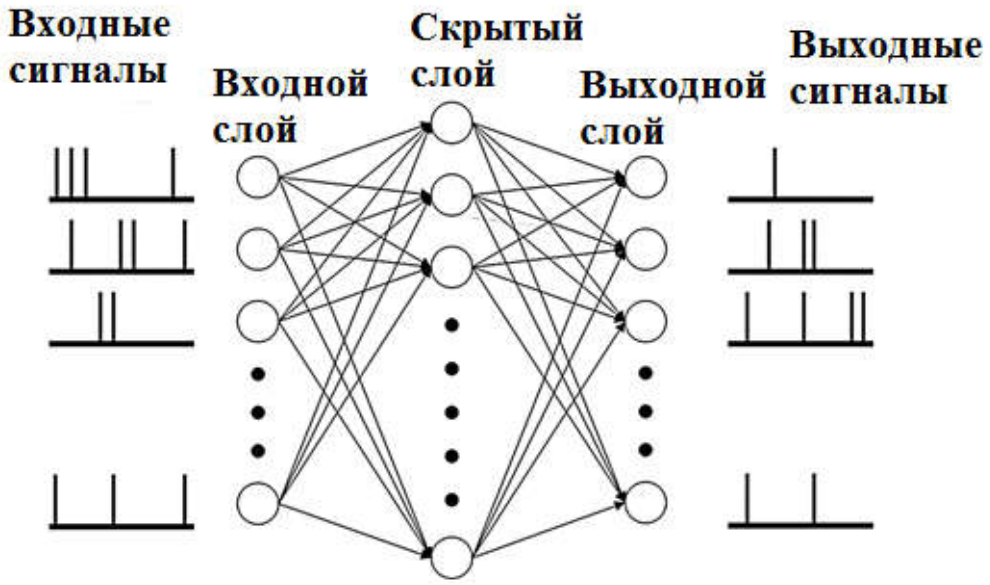


Рисунок 37. Схематичное представление структуры Импульсной нейронной сети

Данный тип сети еще находиться на стадии активного исследования. Тем ни менее ИмНС уже успешно применялась для динамического управления мобильным роботом. В качестве основных ее преимуществ можно назвать:

- Перспективность в обработке динамической информации (например, видеопотока, временных рядов и т. п.).
- Более плотное и эффективное кодирование информации по сравнению с MLP.
- Более эффективное расходование электроэнергии при физической реализации.
- Высокая степень эффективности параллельного выполнения при физической реализации.

К недостаткам можно отнести слабую изученность, сложность математической модели и вычислительную сложность при выполнении на обычном железе. А главный недостаток заключается в отсутствии хорошего алгоритма обучения.

Причины бурного развития ИНС сегодня

Казалось бы, раз ИНС такой мощный инструмент\метод и первая нейронная сеть была предложена так давно, то почему только сейчас появляется столько технологий на нейронных сетях? Тому есть две причины:

Чтобы исследовать глубокие ИНС, нужно проводить эксперименты, а они могли длиться по месяцу, а если и меньше, то нужно было все равно проводить десятки экспериментов. Таким образом, исследовать сети было сложно.

Поэтому появление соответствующих параллельных вычислителей и общее увеличение мощности компьютеров изменило эту ситуацию. Кроме того, появились обычные видео карты (GPU), на которых любой мог запускать сложные вычисления. Поэтому появилось много исследователей, которые проводили эксперименты. То есть, как ни странно, развитие тормозилось не из-за принципиальных возможностей ИНС, а из-за того, что экспериментировать было долго.

Функция активации «Relu» упростила функцию сетей в целом и сделала вычисления Глубоких сетей еще быстрее.

Психологический фактор. Извилистая история нейронных сетей и долгое пребывание в состоянии заморозки создали дурную славу нейронным сетям, и исследователи очень неохотно занимались этой сферой.

Возрастание интереса сразу сложило разные кусочки мозаики, и Сверточные сети получили вторую жизнь, а их успех дал еще больший толчок.

Стоит также отметить, что сегодня наблюдается обратный эффект. Пресса и Интернет часто преувеличивают возможности ИНС или допускают неточности, из-за чего может сложиться впечатление, что нейронные сети – это искусственный интеллект.

Борьба с переобучением в ИНС

Тема переобучения была подробно рассмотрена ранее, поэтому просто перечислим способы борьбы с этим эффектом, подходящие для большинства типов ИНС:

- Кросс-валидация. Данный метод не зависит от модели, это общий подход к тренировке моделей, помогающий контролировать и бороться с переобучением особенно на малых выборках.
- Уменьшение числа слоев\нейронов. Уменьшение слоев и нейронов явно влияет на сложность модели, однако необходимо искать компромисс между обобщающей способностью нейронов и степенью абстрактности признаков.
- Добавление L2 регуляризации. Аналогично Регрессии этот дополнительный метод штрафует высокие веса модели.
- Локальность восприятия. Задавая рецептивное поле, восприятие нейрона можно также уменьшить не только количество параметров или сложность модели, но, и более того, – определить влияющие факторы, что позволит нейрону решать локальные задачи и складывать их в глобальное решение. Это крайне важная особенность ИНС.

Обратное распространение ошибки

Итак, как и у любой другой модели в машинном обучении, у ИНС есть 2 этапа работы:

- обучение;
- использование, моделирование или получение отклика, прогноза (все это подразумевает расчет выхода по обученной модели).

При этом использование сети называют прямым распространением (forwardpropagation), потому что сигнал со входа сети распространяется к выходу (рисунок 38), то есть слева направо.

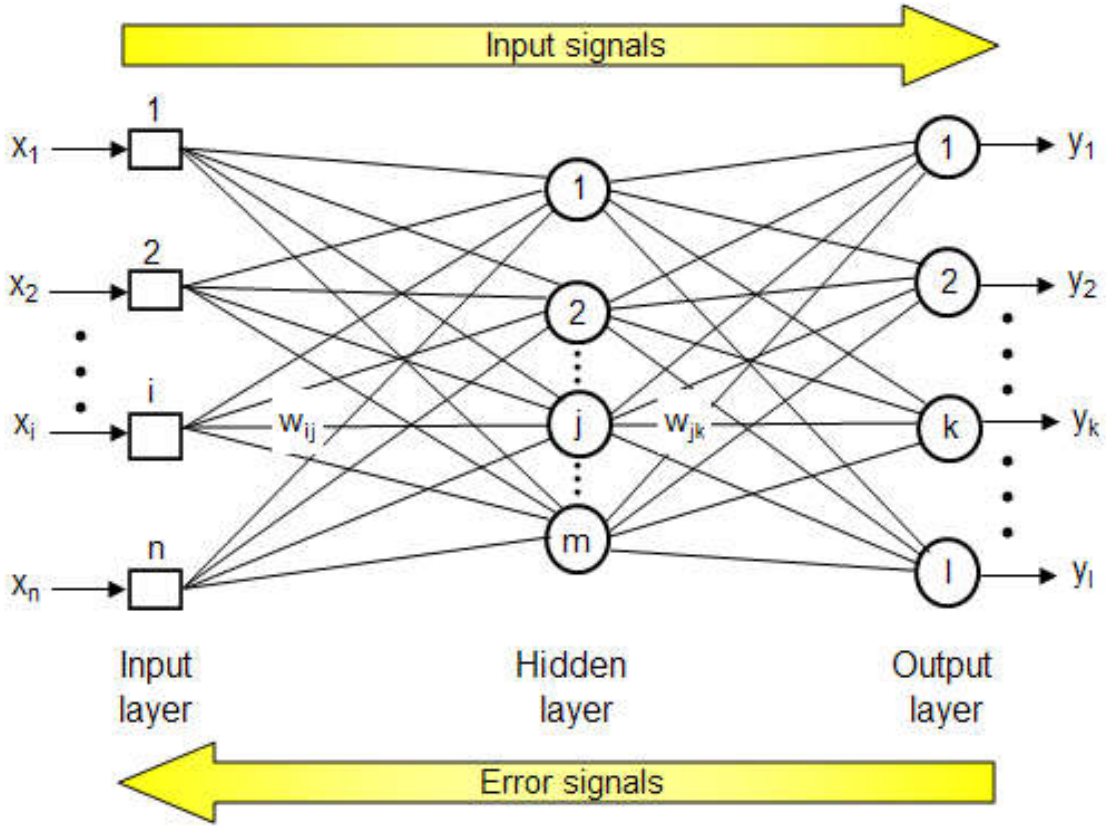


Рисунок 38. Обобщенная схема распространения сигналов по MLP

На рисунке 39 изображена принципиальная схема прямого распространения сигнала со входа на один нейрон. В случае нескольких нейронов в слое схожие вычисления будут проводиться для каждого нейрона. После чего сигнал двинется к следующему слою, и если в нем также несколько нейронов, то там будут сходные вычисления

для каждого нейрона и так далее. Каждый нейрон берет взвешенную сумму своих входов, считает функцию активации и выдает выход для следующего слоя и т. д.

Обучение называют обратным распространением (backpropagation) ошибки, потому что информация идет наоборот, с выхода сети, и информация эта – об ошибке, а не о входном образе, как при прямом распространении.

В модель нейрона на рисунке 39 включен пороговый элемент (bias), который обозначен символом b_k. Эта величина отражает увеличение или уменьшение входного сигнала, подаваемого на функцию активации. По сути, это свободный коэффициент при некотором признаке в нулевой степени (как и у Регрессии).

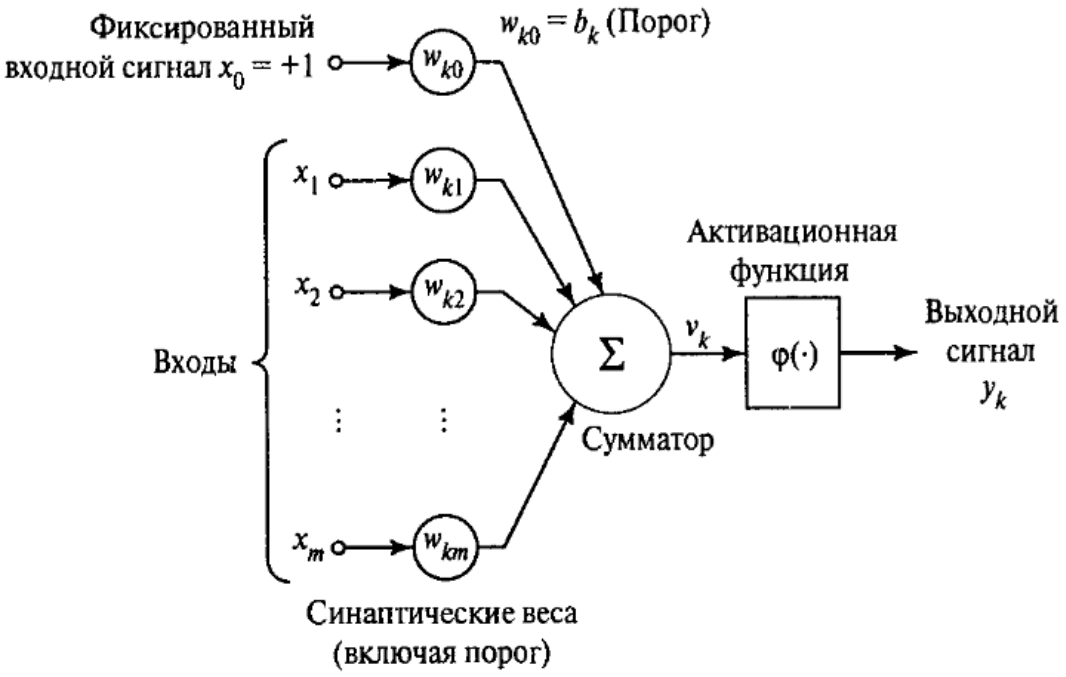


Рисунок 39. Принципиальная схема прямого распространения сигнала

В математическом представлении функционирование нейрона k можно описать следующей парой уравнений:

$U_k = \sum_i^m w_{ki} x_i$,

$y_k = \varphi(U_k)$,

где U_k – индуцированное локальное поле, или потенциал активации.

Серия таких уравнений, вычисленных последовательно, в конце концов даст результат всей сети. Если в последнем слое сети один нейрон, то соответственно получим скаляр или число, а если нейронов несколько, то получим вектор значений для каждого входного образа X.

Теперь рассмотрим алгоритм обратного распространения ошибки для обучения искусственной нейронной сети. Он разработан в первую очередь для сетей MLP-типа. Но также подходит для сверточных сетей, автокодировщиков и, при определенных модификациях, для рекуррентных сетей.

Для начала введем определения:

Обучить сеть – минимизировать ошибку, а именно минимизировать разницу между выходом сети (реакцией на вход) и требуемой реакцией, то есть это обучение с учителем.

Минимизация ошибки производится путем итеративных правок\корректировок весов ИНС. Корректируются в ИНС только веса и больше ничего.

Минимизация ошибки происходит для каждого отдельно взятого входного образа. Повторяя такую минимизацию для каждого образа много раз, получаем общую минимизацию. Количество проходов по тренировочной выборке называется эпохами.

Начнем рассмотрение алгоритма с последнего (выходного) слоя сети (рисунок 40).

Сперва предположим, что выходной слой содержит только один нейрон. Тогда:

E, e – ошибка выхода сети для текущей пары (X, Y);

Y, y – требуемый выход для соответствующего входного образа X;

o = φ(U) – выход нейрона, рассчитывается как сигмоида от U;

U – результат взвешенной суммы, или индуцированное локальное поле;

Wij – вес, соединяющий некоторый нейрон слоя i с некоторым нейроном слоя j.

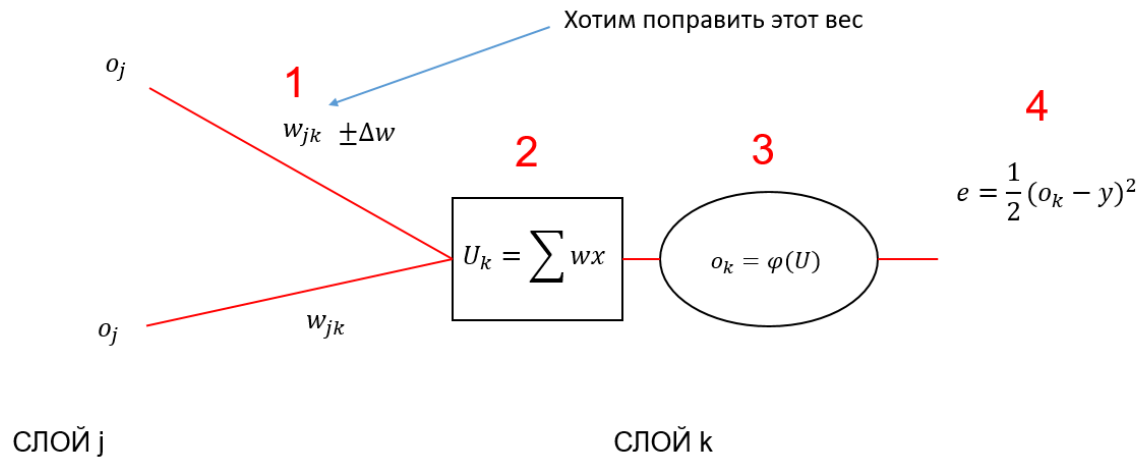


Рисунок 40. Представлена схема последнего слоя сети MLP

Индексы при символах означают индексы слоев. Так как в каждом слое имеется несколько нейронов, то о каком нейроне идет речь, когда пишется oj? О любом произвольном в этом слое, так как формула обобщенная и не вносит конкретики о номере нейрона внутри слоя.

Первая идея алгоритма\метода заключается в том, что мы двигаемся по градиенту ошибки, как и во всех численных градиентных методах. Мы не знаем, где точное решение, но если итеративно понемногу уменьшать ошибку для каждого входного образа, то мы рано или поздно придем к некому равновесному состоянию. Не обязательно минимуму ошибки, но к такому моменту, когда дальнейшие правки уже не будут коренным образом менять ситуацию, а система войдет либо в полную заморозку (изменений вообще не будет), либо в некий колебательный процесс около некоторой матрицы весов.

Собственно, двигаясь по антиградиенту ошибки $-\nabla F$, мы как раз и производим корректировки весов на некие дельта (в данном случае у весов нет никаких индексов, потому что это общая концепция правок):

$$w^{t+1} = w^t - \alpha\Delta w,$$
$$w^{t+1} = w^t - \alpha\nabla F(w^t),$$
$$\nabla F(w^t) = \frac{\partial E}{\partial w^t},$$

где w^{t+1} – вес в следующий момент времени;

w^t – вес в текущий момент времени;

Δw – правка веса на текущем шаге, в сторону уменьшения ошибки на текущем шаге;

α – размер шага, скорость движения по антиградиенту или сила правок весов.

Еще раз о том, почему используется именно градиент. Потому что нельзя рассчитать точное значение правок для всех весов сразу, ведь мы не знаем вклад каждого веса в ошибку, мы лишь знаем значение ошибки и направление. Чтобы понять суть идеи, представьте, что некая правка $\pm\Delta w$ некого веса w даст нам увеличение или уменьшение ошибки на выходе сети (см. рисунок 40). Получается, что небольшие правки весов приводят к некоторым небольшим изменениям конечной ошибки сети. Возможно ли найти функциональную зависимость между этими небольшими изменениями? Что есть отношение небольшого изменения ошибки к изменению какого-либо веса? Это производная $\frac{\partial E}{\partial w}$.

Таким образом, найдя аналитическое выражение производной ошибки по любому весу, мы определим характер воздействия правок этого веса на ошибку. А в конкретных точках этой функции (при конкретном входном образе и всех других параметрах сети) мы сможем точно подсчитать значение функции производной, то есть сможем оценить знак и значение dE, на которое будет меняться ошибка при наших правках dw. А значит, мы можем выбрать такую правку, чтобы уменьшить эту самую E. При этом все остальные веса, кроме того который правится в данный момент, замораживаем.

Итак, для последнего уровня все относительно просто. Чтобы найти производную $\frac{\partial E}{\partial w}$, необходимо взять серию производных по каждой вложенной функции (по цепному правилу дифференцирования):

$$w_{jk}^{t+1} = w_{jk}^t - \alpha\frac{\partial E}{\partial w_{jk}} =$$

$$= w_{jk}^t - \alpha\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial U_k}\frac{\partial U_k}{\partial w_{jk}} = w_{jk}^t - \alpha(o_k - y)o_k(1 - o_k)o_j.$$

Но что делать с внутренними слоями? Дело в том, что для нейронов внутренних слоев мы не можем явно рассчитать не то что значение правки, но и значение ошибки. Ведь чтобы рассчитать значение ошибки, надо знать, какое значение выхода должно быть у каждого нейрона внутреннего слоя. А мы не можем этого знать, ведь не знаем конкретный вклад каждого нейрона в ошибку. Мы знаем производную, то есть ближайший характер изменений ошибки от правок весов, но не можем посчитать нужные значения для произвольных точек.

Но что если пойти таким же путем, как и раньше? Допустим, что некая правка $\pm\Delta w$ некого веса w (теперь уже на скрытом слое) даст нам увеличение или уменьшение ошибки на выходе сети (см. рисунок 41). Получается, что небольшие правки этого веса тоже приводят к некоторым небольшим изменениям конечной ошибки сети при условии, что все остальные веса сети константны в рамках этого временного среза.

Рисунок 41. Схема последних двух слоев сети

Поэтому можно взять производную выходной ошибки по этому нейрону. Это будет такая же цепочка производных (только вес w_{ij} теперь будет константой, а не переменной, поэтому он и останется после взятия производной по o_j, а берем мы именно по o_j, чтобы пройти (протиснуть информацию) дальше).

$$w_{ij}^{t+1} = w_{ij}^{t} - \lambda\frac{\partial E}{\partial w_{ij}} = w_{ij}^{t} - \lambda\frac{\partial E}{\partial o_k}\frac{\partial o_k}{\partial U_k}\frac{\partial U_k}{\partial o_j}\frac{\partial o_j}{\partial U_j}\frac{\partial U_j}{\partial w_{ij}}.$$

Теперь можно подставить каждую производную и получить формулу корректировки веса в скрытом слое. Однако выше мы делали серьезное допущение, что в последнем слое будет только один нейрон. А это может быть не так. И нас интересует именно этот общий случай и для любого слоя. Таким образом, из этой формулы надо выделить закономерность, которую можно использовать для произвольного слоя:

$$w_{ij}^{t} - \lambda \left| \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial U_k} \frac{\partial U_k}{\partial o_j} \right| \frac{\partial o_j}{\partial U_j} \left| \frac{\partial U_j}{\partial w_{ij}} \right| .$$

1 2 3

В формуле градиента ошибки по весу скрытого слоя присутствуют 3 составляющих. 1 – это составляющая следующего слоя, 2 – это составляющая текущего слоя и 3 – составляющая предыдущего слоя. То есть изменение выходной ошибки складывается из влияния этих составляющих, если все остальные элементы среды заморозить. А раз так, то можно сделать два вывода:

Первая составляющая в случае нескольких нейронов в следующем слое будет не одна. Понятно, что текущий нейрон j-го слоя распространяет свою ошибку на все связанные с ним нейроны слоя k. А значит, надо просуммировать вклад, который оказывает корректировка веса по всем путям. Или, иными словами, просуммировать производные по разным связям нейронов. Тогда первый блок можно будет переписать так:

$$\sum_{k=0}^{M} \frac{\partial E}{\partial o_k} \frac{\partial o_k}{\partial U_k} \frac{\partial U_k}{\partial o_j} = (o_k - y)\varphi'_k w_{jk}.$$

Для сколь угодно далекого слоя можно не считать полный градиент, а брать вклад следующего слоя за основу. Ведь любой произвольный слой в первую очередь делает вклад именно в следующий слой. Таким образом, учитывая промежуточные вклады или точнее влияние всех промежуточных вкладов, мы сможем оценить итоговое влияние на ошибку. Опять же речь идет о небольших локальных приращениях $\pm\Delta w$ по конкретному w. Значит, можно выделить первую составляющую формулы в отдельную передаточную

величину вклада ошибкиδ . Также ее называют производной ошибки по взвешенной сумме, по локальному индуцированному полю (по U). Тогда для последнего слоя:

$\delta_k=(o_k - y)o_k(1 - o_k)$.

А для любого другого скрытого слоя:

$\delta_j=(\sum \delta_k w_{jk})o_j(1 - o_j)$.

Получается, что сигма рассчитывается как совокупность всех вкладов в ошибку (начиная с конца). И именно эта величина является той информацией, которая выражает ошибку от слоя к слою и распространяется обратно. Учитывая оба пункта одновременно, формулы для корректировок можно переписать следующим образом:

$\Delta w = -a\delta_k o_j = -a(o_k - y)o_k(1 - o_k)o_j$ – если корректируется вес последнего слоя.

$\Delta w = -a\delta_j o_i = a(\sum \delta_k w_{jk})o_j(1 - o_j)o_i$ – если корректируется вес скрытого слоя.

Общий алгоритм можно представить так:

- Инициализировать веса случайным образом;
- Рассчитать выход сети прямым распространением сигнала;
- Рассчитать ошибку на выходе;
- Рассчитать корректировки весов текущего слоя по ошибке следующего;
- Обновить веса.

Условия завершения алгоритма обратного распространения ошибки:

- Требуемая величина ошибки;
- Максимальное количество итераций;
- Последние N проходов веса не правились больше чем на T.

Замечание 1: шаг градиентного спуска или скорость градиентного спускаaкрайне важный параметр и нужен не только ради формальности. Дело в том, что при $a = 1$движение по градиенту ошибки будет слишком быстрым, т.к. каждая корректировка отдельно взятого веса производиться с учетом заморозки всех остальных весов.

А следующий вес корректируется на основании исходного значения ошибки (вычисленного еще до правки первого веса). В противном случае движение по градиенту будет плавным, но это очень расточительно с точки зрения вычислительных ресурсов (такой алгоритм будет очень медленным). Кроме того, точный пересчет в рамках одного образа и не нужен, так как полное уменьшение ошибки для одного образа не гарантирует факт того, что уменьшиться общая ошибка. А уменьшить необходимо именно общую ошибку, а значит в рамках одного образа корректировки должны быть небольшими. Уточним, что на практике часто принимают $a = 0.01$.

Замечание 2: конечно, рассмотренный алгоритм не лишен недостатков. Некоторые из них очевидны и их возможно устранить. Существует множество модификаций градиентного спуска, например: Adadelta, Adam и т. п. Есть и другие методики улучшения обучения. Один из самых мощных и простых – это Batching. Но все эти методы выходят за рамки пособия, так как при необходимой фундаментальной подготовки вы можете изучить их особенности самостоятельно.

Замечание 3: алгоритм обратно распространения ошибки линеен с точки зрения вычислительной сложности.

Замечание 4: более подробное рассмотрение модификаций алгоритма обратного распространения выходит за рамки этого пособия, как и рассмотрение альтернативных функций активации (таких как ReLU, LeakyReLU, Parametric ReLU, RandomizedReLU). Поэтому вам предлагается изучить эти вопросы самостоятельно.

Нечеткие нейронные сети

Нечеткой нейронной сетью (НС) обычно называют четкую нейросеть, которая построена на основе многослойной архитектуры с использованием специальных «И»-, «ИЛИ»-нейронов.

Нечеткая нейросеть функционирует стандартным образом на основе четких действительных чисел, нечеткой является только интерпретация результатов.

Нечеткие нейронные сети осуществляют выводы на основе аппарата нечеткой логики, а параметры функций принадлежности настраиваются с использованием алгоритмов обучения НС. Поэтому для подбора параметров таких сетей применим метод обратного распространения ошибки, изначально предложенный для обучения многослойного персептрона. Нечеткая нейронная сеть, как правило, состоит из четырех слоев: слоя фаззификации входных переменных, слоя агрегирования значений активации условия, слоя агрегирования нечетких правил и выходного слоя.

Наибольшее распространение в настоящее время получили архитектуры нечеткой НС вида ANFIS и TSK [18]. Доказано, что такие сети являются универсальными аппроксиматорами. Быстрые алгоритмы обучения и интерпретируемость накопленных знаний – эти факторы сделали сегодня нечеткие нейронные сети одним из самых перспективных и эффективных инструментов мягких вычислений.

Преимущества нечетких нейронных сетей

Основным преимуществом технологии нейрокомпьютинга служит возможность выразить зависимость «выход-вход» без предварительной аналитической работы по выявлению правил, а на основе обучения на примерах. Недостатком нейросетей является невозможность объяснить выходной результат, так как значения распределены по нейронам в виде значений коэффициентов весов. Основной трудностью в применении нечетких экспертных систем служит необходимость явно сформулировать правила проблемной области в форме правил. В нечетких экспертных системах легко построить объяснение результата в форме протокола рассуждений. Поэтому в настоящее время создаются гибридные технологии.

Примером гибридной технологии служит реализация базы нечетких правил на основе нейросети. База нечетких правил для двух входных переменных имеет следующую структуру:

R_i : *if* x_{1i} *is* A_{1i} *and* x_{2i} *is* A_{2i} *then* z_i *is* C_i.

Простой реализацией базы нечетких правил служит интерпретация базы правил как таблицы определения некоторой функции, то есть базу правил можно представить обучающей выборкой:

$$\{((A_{1i}, A_{2i}), C_i)\}.$$

Например, обучающая выборка в нечетких терминах может быть сформулирована следующим образом:

{((малое, большое), около нуля)}.

Чтобы совместить две технологии: технологию нечетких систем и технологию нейрокомпьютинга, необходимо предложить способ четкого дискретного представления непрерывных функций принадлежности. Чтобы представить в четких данных непрерывные функции принадлежности, выберем максимально большой интервал $[x_1,x_2]$, в котором представлены все нечеткие множества условных частей правил. Разбиваем интервал с равным шагом, тогда любое нечеткое значение представляется четким вектором. Другой способ представления нечеткого понятия в вид четких данных состоит в представлении нечеткого множества в виде совокупности α-срезов.

α-срезом называется четкое множество, включающее все элементы *x* некоторого нечеткого множества *X*, принадлежность которых больше равна α.

$$\mu_X(x) \geq \alpha .$$

Каждое α-подмножество представляется двумя числами – левой и правой границей α^L, α^R, то есть α-срезы четко представляют непрерывную функцию принадлежности.

Изменение элемента нейросети для адаптации к нечетким системам может касаться выбора функции активации, реализации операций сложения и умножения, так как в нечеткой логике сложение моделируется любой треугольной конормой (*max, a + b – a×b...*), а операция умножения треугольной нормой (*min, a×b ,...*).

И-нейроном (AND-нейроном) называется нейрон, в котором умножение веса на вход моделируется конормой *S(w,x)*, а сложение – нормой *T(w,x)*.

ИЛИ-нейроном (OR-нейроном) называется нейрон, в котором умножение веса и входа моделируется нормой $T(w,x)$, а сложение взвешенных весов конормой $S(w,y)$ $Y = S(T(w_1,x_1),T(w_2,x_2))$.

Пример: $(max(min(w_1,x_1),min(w_2,x_2)))$.

В качестве функции активации обычно используют функцию

$F(x) = 1/(1+exp(b(x-a)))$.

Нечеткой нейросетью называют четкую нейросеть, которая построена на основе многослойной архитектуры с использованием «И-», «ИЛИ-нейронов».

Нечеткая нейросеть функционирует стандартным образом на основе четких действительных чисел. Нечеткой является только интерпретация результатов. При создании гибридной технологии, кроме объединения систем по данным, можно использовать нейрокомпьютинг для решения частной подзадачи нечетких экспертных систем, а именно настройки параметров функции принадлежности. Функции принадлежности можно сформировать двумя способами: методом экспертной оценки или на основе статистики. Гибридные технологии предлагают третий способ: в качестве функции принадлежности выбирается параметризованная функция формы (например, параметризованная Гауссова кривая), параметры которой настраиваются с помощью нейросетей. Настройка параметров может быть получена в рамках алгоритма обратного распространения ошибки.

Пусть задана следующая система нечетких правил:

If x_1 *is* A_{1i} *and ...* x_n *is* A_{ni} *then* z_i *is* C_i ,

где A_i – нечеткие числа; C_i– действительные числа. Значение $\alpha_j=\Pi_i\alpha_i$ – сила или достоверность правила, $i = 1,..,n$ – номер входной переменной, $j = 1,..,m$ – количество правил.

$$Z = \frac{\sum_{j=1}^{m} a_j z_j}{\sum_{j=1}^{m} a_j},$$

где Z – вычисленное значение выхода.

Допустим, что разработана нейросеть с n входами и одним выходом. Каким образом такая НС может аппроксимировать базу нечетких правил? Любая совокупность нечетких правил может рассматриваться как нелинейное соответствие, заданное таблицей определения $\{(x,y)\}$, где x – вектор входа; y – желаемое значение выхода; а z – значение, вычисляемое нейросетью. Тогда можно определить текущую ошибку:

$E^K=1/2\times(z^K-y^K)^2$.

То есть можно применить стандартный алгоритм коррекции ошибки на основании данного определения

$Z(t+1) = Z(t)- \zeta\times (\partial E^K/\partial Z)$,

где ζ – уровень обучения; $\partial E^K/\partial Z$ – направление градиента снижения ошибки.

$Z(t+1) = Z(t)- \zeta\times (Z^K-Y^K) \times \alpha_j / (\alpha_1 +\alpha_2 +...+\alpha_m)$.

При применении стандартного алгоритма обратного распространения ошибки для того, чтобы настроить выход, необходимо изменить параметры функции принадлежности условных частей, то есть обучение сети позволит настроить функцию принадлежности с точки зрения обучающей выборки. При практической реализации системы нечетких правил важным является вопрос о типичных представителях нечетких значений в правилах. Большинство нечетких понятий, представленных лингвистическими переменными, выражает свои значения с помощью количественных нечетких множеств: NB – отрицательное большое; NM – отрицательное среднее; NS – отрицательное малое; ZE – около нуля; PS – положительное малое; PM – положительное среднее; PB – положительное большое.

Пример использования нечеткой нейронной сети

Рассмотрим нечеткие нейронные сети на примере нечеткого регулятора для стиральной машины [18], то есть построим fuzzy-neuro контроллер. Выберем для демонстрации технологии нечетких нейронных сетей архитектуру ANFIS (Adaptive Network Based Fuzzy Inference System). Основа интеграции нейронных сетей и систем нечеткого вывода заключается в том, что оба метода представляют

нелинейное отношение в пространстве входы-выходы. Важная задача нечеткого моделирования – это настройка функций принадлежности, являющаяся по существу задачей оптимизации. Как нейронные сети, так и генетические алгоритмы используются для ее решения. Самый простой подход требует назначить определенную параметризованную функцию формы в качестве функции принадлежности и подобрать параметры на основе обучения нейронной сети. Рассмотрим простой пример с тремя нечеткими правилами вывода в базе знаний.

R_1: ЕСЛИ <количество_белья> is <много>

И <температура_воды> is <высокая> И <загрязненность> is <высокая>

ТО <длительность> is <высокая>;

R_2: ЕСЛИ <количество_белья> is <много>

И <температура_воды> is <высокая> И <загрязненность> is <низкая>

ТО <длительность> is <низкая;

R_3: ЕСЛИ <количество_белья> is <мало>

И <температура_воды> is <низкая> И <загрязненность> is <низкая>

ТО <длительность> is <малая>;

Зададим следующие функции формы для высказываний. Пусть высказывание <количество_белья> is <мало> соответствует функции $L_1(x)$ и высказывание <количество_белья> is <много> – функции $H_1(x)$.

$L_1(x)= 1/(1+exp(b_1(x-c_1))$,

$H_1(x)= 1/(1+exp(-b_1(x-c_1))$,

причем $L_1(x) + H_1(x) =1$.

Аналогично для высказывания <температура_воды> is <высокая> будем использовать функцию $L_2(t)$ и для <температура_воды> is <низкая> – функцию $H_2(t)$.

$L_2(t)= 1/(1+exp(b_2(t-c_2))$,

$H_2(t)= 1/(1+exp(-b_2(t-c_2))$,

причем$L_2(t) + H_2(t) =1$.

Для высказывания <загрязненность> is <высокая> определим функцию $L_3(z)$ и для <загрязненность> is <низкая> – функцию $H_3(z)$.

$L_3(x)= 1/(1+exp(b_3(z-c_3))$,

$H_3(x)= 1/(1+exp(-b_3(z-c_3))$,

$L_3(z) + H_3(z) =1$.

Для выходов <длительность> is <высокая> и <длительность> is <малая> аналогично определим функции

$L_4(y)= 1/(1+exp(b_4(y-c_4))$,

$H_4(y)= 1/(1+exp(-b_4(y-c_4))$,

то есть $L_4(x) + H_4(x) =1$.

Для четких значений <количество_белья>, <температура_воды> и <загрязненность>: A_1, A_2, A_3 определим релевантность (силу) правил α:

$\alpha_1 = H_1 \wedge H_2 \wedge H_3$,

$\alpha_2 = H_1 \wedge H_2 \wedge L_3$,

$\alpha_3 = L_1 \wedge L_2 \wedge L_3$.

Выходы по каждому из правил определяется с помощью обратных функций принадлежности правых частей правил.

$Y_1 = H_4^{-1}(\alpha_1)$,

$Y_2 = H_4^{-1}(\alpha_2)$,

$Y_2 = L_4^{-1}(\alpha_3)$.

Общий выход из системы нечетких правил определяется как

$y_0 = (\alpha_1 \times y_1 + \alpha_2 \times y_1 + \alpha_3 \times y_1)/(\alpha_1 + \alpha_2 + \alpha_3)$.

Далее построим нечеткую нейронную сеть, идентичную системе нечеткого вывода, и обучим функции принадлежности анцедента и консеквента правил (рисунок 42).

Слой 1. Выходы узлов – это степени, в которых заданные входы удовлетворяют функциям принадлежности, ассоциированным с этими узлами.

Слой 2. Каждый узел вычисляет силу правила. Выход верхнего нейрона $\alpha_1 = H_1 \wedge H_2 \wedge H_3$, выход среднего нейрона $\alpha_2 = H_1 \wedge H_2 \wedge L_3$, а выход нижнего – $\alpha_3 = L_1 \wedge L_2 \wedge L_3$. Все узлы помечены Т, так как

можно выбрать любую Т-норму для моделирования логического И. Узлы этого слоя называются узлами правил.

Слой 3. Каждый узел помечен N, чтобы показать, что узлы нормализуют силу правил $\beta_i = \alpha_i / (\alpha_1 + \alpha_2 + \alpha_3)$.

Слой 4. Выход нейронов – это произведение нормализованной силы правила и индивидуального выхода соответствующего правила.

$\beta_1 Y_1 = \beta_1 H_4^{-1}(\alpha_1)$,

$\beta_2 Y_2 = \beta_2 H_4^{-1}(\alpha_2)$, $\beta_3 Y_2 = \beta_3 L_4^{-1}(\alpha_3)$.

Слой 5. Одиночный выходной нейрон вычисляет выход сети

$y_0 = \beta_1 Y_1 + \beta_2 Y_2 + \beta_3 Y_2$.

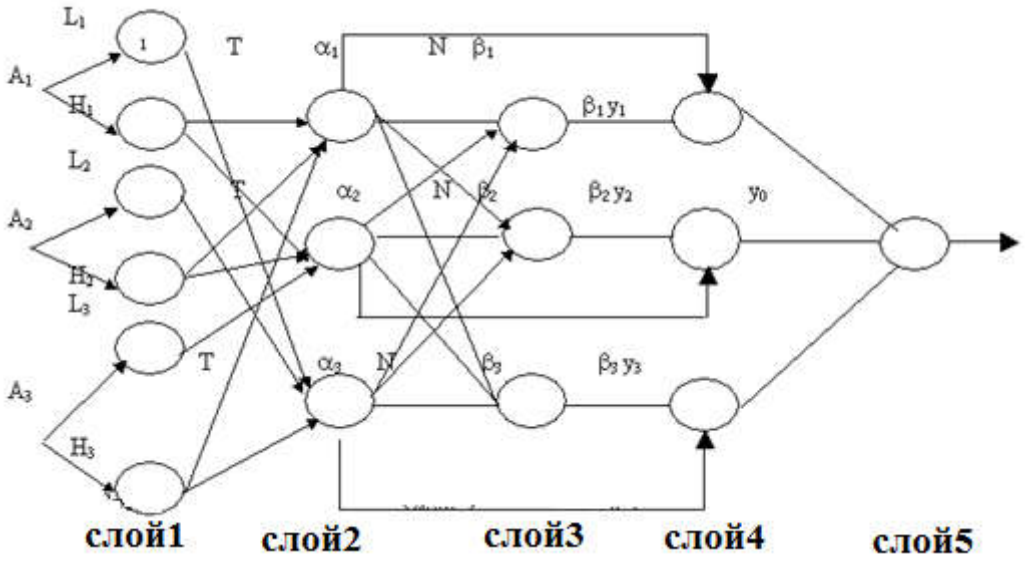


Рисунок 42. Пример нечеткой нейронной сети

Алгоритм обучения для нечеткой нейронной сети примера

Пусть задана четкая обучающая выборка

$\{(x_1,t_1, z_1 y_1), \ldots (x_k, t_k, z_k y_k)\}$, где x,t,z – входные условия количества белья, температуры воды и загрязненности, а y – длительность стирки. Ошибку на k-м образце определим как обычно:

$E^k = 1\backslash 2(O_i - Y_i)^2)$.

Используем традиционный градиентный метод для обучения параметров левой и правой частей нечетких правил. Покажем, как можно настроить параметры функции формы.

$b_4(t+1) = b_4(t) - \zeta \times (\partial E/\partial b_4)$,

$c_1(t+1) = c_1(t) - \zeta \times (\partial E/\partial c_1)$

.....

$c_4(t+1) = c_4(t) - \zeta \times (\partial E/\partial c_4)$.

Используя данные соотношения как правила изменения весов в алгоритме обратного распространения ошибки, можно настроить параметры функций принадлежности в ходе обучения нечеткой нейронной сети.

Генетические алгоритмы

Генетические алгоритмы представляют собой адаптивные методы поиска, использующиеся для решения задач функциональной оптимизации. В их основе лежит идея природной эволюции, когда популяции развиваются в течение нескольких поколений и подчиняются законам естественного отбора. Лучшая особь выбирается по принципу «выживает наиболее приспособленный». Как мы помним, он был открыт Чарльзом Дарвином. Согласно этим принципам генетические алгоритмы способны «развивать» решения реальных задач при соответствующем формальном описании. Причем в отличие от эволюции, генетические алгоритмы моделируют лишь существенные для развития процессы.

В природе действует такой механизм, что наиболее приспособленные особи имеют больше шансов на воспроизведения потомства. Причем комбинация наиболее хороших характеристик может привести к появлению максимально приспособленного потомка. Генетические алгоритмы используют прямую аналогию с этим механизмом. Множество возможных решений проблемы – это популяция. Каждое решение оценивается по степени его «приспособленности» для решения поставленной задачи.

Сфера применения генетических алгоритмов весьма обширна. Они могут использоваться при создании таких вычислительных структур, как автоматы или сети сортировки. В машинном обучении их

можно использовать для проектирования нейронных сетей или в управлении роботами. Их можно использовать для моделирования процессов в различных областях (биологические, социальные, когнитивные системы). Однако наиболее популярное приложение – оптимизация многопараметрических функций, когда нужно найти оптимальное значение, зависящее от некоторых входных параметров.

В своей работе генетический алгоритм работает с хромосомами, которые состоят из генов. Чаще всего ген – это 0 или 1, говорящая о включении или не включении признака, характеризующего решение, в хромосому. Соответственно, хромосома – это битовая строка, описывающая решение. Но иногда возможны и другие кодировки в зависимости от задачи.

При работе с генетическим алгоритмом нам необходимо определить следующее:

Кодировку хромосомы (что будет представлять собой ген, и как гены будут участвовать в характеристике решения).

Пространство гипотез (популяцию), из которых мы должны выбрать лучшую.

Функцию приспособленности, оценивающую хромосомы.

Набор и вид генетических операций (скрещивание, мутацию).

Критерий остановки алгоритма (либо желаемое оптимальное значение, либо количество шагов эволюции популяции).

Для примера рассмотрим решение задачи с помощью генетического алгоритма. Формулировка задачи: необходимо составить наиболее сбалансированный рацион питания, удовлетворяющий определенным медицинским требованиям. Известен перечень продуктов из N наименований, каждый из которых имеет такие характеристики, как содержание жиров, белков, углеводов, калорий, а также известны рамки допустимого их содержания. Подходящим считается рацион, который лучше всего удовлетворяет определенным медицинским требованиям. Например, если рекомендуемое содержание жиров – 50, белков – 60, углеводов – 70, клетчатки – 80, витамина B – 90, витамина C – 100, то лучшим будет считаться рацион, чье суммарное

отклонение минимально, но чья десятая часть, например, укладывается в рамки от 40 до 80.

Первое, что нам необходимо сделать, – выбрать кодировку хромосомы. Так как основная наша задача – подобрать список продуктов, то хромосомой может быть весь перечень из N продуктов, а 0 или 1 в ней будут говорить о включении или не включении продуктов в рацион. Для лучшего понимания рассмотрим конкретный пример. Пусть у нас есть 4 продукта, каждый из которых можно описать набором из 6 числовых параметров-характеристик. Тогда по сути получим массив чисел:

Список_объектов={Объект$_1$={20,30,40,50,60,70},

Объект$_2$={40, 40, 40, 40, 40, 40},

Объект$_3$={30, 30, 30, 30, 30, 30},

Объект$_4$={20, 30, 40, 40, 30, 30}}.

Тогда хромосома может иметь вид:

1010

Это говорит о том, что в рацион включаются первый и третий объекты. Тогда популяция может иметь вид:

0100

1010

1100

0101

Теперь определим функцию приспособленности для оценки каждой хромосомы в нашей задаче. Пусть у нас есть набор эталонных значений для каждой из характеристик объектов:

Эталон= {50,60,70,80,90,100}.

Тогда функция приспособленности будет иметь вид:

$$F = \sum_{j=1}^{N} \left(\frac{\sum_{i=1}^{m} \text{Эталон}_i - \text{Объект}_{ij}}{10} \right) \times Hrom_j \ ,$$

где $Hrom_j$– соответствующий ген в хромосоме.

Для оценки приспособленности хромосомы необходимо также проверить, укладывается ли значение функции в рамки от 40 до 80 (добавлением соответствующих условий), а для выявления наиболее приспособленной особи необходимо найти минимальную подходящую *F* в популяции.

Обратите внимание!

- *Определение вида функции приспособленности – чрезвычайно важная задача, потому что от правильности ее определения будет во многом зависеть успешность решения.*

Теперь рассмотрим генетические операции мутации и скрещивания. Если говорить формально, то мутация – это изменение одного или нескольких генов хромосомы вследствие случайного влияния. В контексте нашей задачи это может быть принудительное включение некоего продукта в набор, изменение значения вхождения продукта в набор на противоположное, принудительное исключение некоего продукта из набора и т. д. То есть, по сути, мутация – это изменение значения одного или нескольких генов хромосомы на противоположный или четко заданный. Вид и критерий применения мутации выбираются эмпирически. Допустим, для нашей задачи мы решили, что мутация будет изменять случайный ген на противоположный. Тогда, если исходная хромосома имела вид 0010, то мутировавшая может быть такой: 1010.

Операция скрещивания, или кроссовер, – операция, которая получает из двух хромосом одну, используя заданную маску. По сути из каждой хромосомы «вырезается» кусок, который помещается в новую. Существует несколько видов кроссовера. Одноточечный кроссовер: «разрез» хромосомы происходит только в одной точке, и новая особь получается путем соединения первой части первой хромосомы и второй части второй хромосомы. Двухточечный кроссовер: есть две точки «разреза», и новая хромосома получается из двух частей первой хромосомы и одной части второй.

Одноточечный кроссовер:

Хромосома$_1$ = 1001, Хромосома$_2$=1100. Точка разреза = 2. Маска потомка: 1часть_Хромосома$_1$+2часть_Хромосома$_2$. Потомок=(10)(00).

Двухточечный кроссовер:

Хромосома$_1$ = 1001, Хромосома$_2$=1100. Точки разреза = 1,3. Маска потомка: 1часть_Хромосома$_1$ + 2часть_Хромосома$_2$ + +3часть_Хромосома$_1$. Потомок=(1)(10)(1).

Теперь разберемся с таким вопросом, как отбор хромосом для воспроизведения потомства и выживание в популяции. Обычно в популяцию выбирается s наиболее приспособленных особей, которые и дают в ней потомство. Для того чтобы выбрать хромосомы в популяцию, можно использовать разные методы: турнирный, ранговый или метод рулетки. Рассмотрим эти методы подробнее.

Турнирный метод: задаем некую фиксированную вероятность выживаемости хромосомы, обозначив ее p. Случайно выбираем две особи. С вероятностью p выживает наиболее приспособлена, а 1-p – менее приспособленная.

Метод рулетки: вычисляем удельную приспособленность каждой особи относительно суммарной приспособленности популяции. Эту величину используем в качестве значения вероятности выживания.

Ранговый метод: выживает s наиболее приспособленных особей.

Таким образом, общую схему генетического алгоритма можно описать так:

Генерация начальной популяции из N особей.

Оценка приспособленности особей.

Пока не сработало условие выхода, делаем следующее:

Выберем s особей для новой популяции и кроссовера.

Выполним кроссовер и мутацию.

Оценим приспособленность итоговой популяции.

Нечеткие системы с генетической настройкой

Настройка функций принадлежности с помощью нейронной сети или генетического алгоритмы (ГА) устраняет принципиальную слабость теории нечетких систем – субъективность функций принадлежности. Применение нейронных сетей к настройке функций

принадлежности позволяет рассматривать окончательную форму функции как аппроксимацию обучающей выборки. Такую же задачу можно решить с помощью ГА, как метода стохастической оптимизации. Генетической нечеткой системой называют нечеткую систему, функции принадлежности и база правил которой спроектирована с помощью генетического алгоритма.

Применить ГА – это значит выбрать единицу кодирования, то есть хромосому; уточнить эволюционные операторы рекомбинации, мутации и селекции; сформировать функцию адаптивности (fitness-function, performanceindex). В настоящее время ГА используют либо для настройки функций принадлежности (базы данных), либо для формирования базы правил, либо для одновременного формирования и функций принадлежности и правил (а именно, базы знаний). В соответствии с объектами оптимизации выбирают единицу кодирования – хромосому. Для настройки функций принадлежности (ФП) за хромосому выбирают одно правило (Мичиганский подход), для настройки базы правил за хромосому выбирают вариант базы правил (Питтсбургский подход, подход итеративного обучения правил). В соответствии с кодированием уточняются правила генерации новых хромосом. Функция адаптивности представляет собой механизм нечеткого вывода, который для каждого варианта базы правил строит либо управление для нечеткого контроллера, либо экспертное заключение для диагностической экспертизы. Как видно из механизма нечеткого вывода, все нечеткие правила вносят вклад в окончательный результат, то есть правила сотрудничают. Но при отборе правил (хромосом) для генерации новых правил ГА накапливает правила, внесшие максимальный вклад в общий результат, то есть хромосомы конкурируют. В этом случае имеет место проблема «конкуренции и кооперации» в генетических нечетких системах (acompetition vs. a cooperation). Решение проблемы в каждом конкретном случае гене-тической нечеткой системы (ГНС) строится эвристически, например, при подходе итеративного обучения правил используют два этапа оптимизации. На первом шаге правила конку-

рируют за право войти в базу правил, а на втором – взаимодействуют при формировании общего результата.

Нечеткие нейронные сети с генетическим проектированием

Рассмотрим возможности генетических вычислений как средства структурной оптимизации нечетких нейронных сетей. Генетические вычисления можно применить на этапе проектирования нейронной сети. Возможности нейронных сетей интерполировать значения временных рядов могут быть широко использованы для оценки поведения макроэкономических показателей, в том числе индексов деловой активности или уровня ценных бумаг. Задача построения нейронного предиктора связана с принятием решений во многих точках проектирования. Необходимо исследовать входные данные и решить, по какому отрезку входных данных рационально делать предсказания, сколько точек в будущем разумно предсказать. Пространство перебора решений организации входных и выходных данных огромно для развитых рынков ценных бумаг, накопивших статистические сведения за десятки лет. Следующей точкой решения служит выбор типа нейронной сети: многослойный персептрон, радиально базисная сеть, вероятностная нейронная сеть, сеть регрессии и т. д. Для выбранного типа НС необходимо определить количество скрытых слоев, количество нейронов в них, виды функций активации и другие. Для каждой сети необходимо выбрать алгоритм обучения и его параметры, например, использование моментов (уровень моментов), уровень обученности, целевой уровень накопленной ошибки и т. д. Для нечеткой нейронной сети необходимо определить архитектуру сети, параметры функций принадлежности. В результате пространство решений при формировании нейронной сети становится необозримым для исследователя, и целесообразно применить ГА как средство эволюционного проектирования.

Генетическая оптимизация F-преобразования временных рядов

Задачи переборного типа с меньшими затратами могут быть решены путем применения эволюционных алгоритмов. Предлагается следующий классический алгоритм генетической оптимизации:

Имеется функция $F(x_1, \ldots, x_4)$ – оценка построения прогноза тренда (должна быть минимизирована), где

x_1 – степень авторегрессии при построении прогноза тренда;

x_2 – метод, которым производится прогноз;

x_3 – количество точек, покрываемых базисной функцией;

x_4 – прогноз на основании истории из предыдущих N точек.

Для выявления сезонности следует строить прогноз, отодвигая историю на t точек назад:

$$F_k = \alpha F_{k-t} + \beta F_{k-t-1} + \ldots$$

Алгоритм генетической оптимизации

1. Хромосомы имеют битовое представление (кодируем в коде Грея для получения отличия соседних хромосом в 1 бите). При этом в начале производится нормировка значений параметров к интервалу [0, 1].
2. Оператор скрещивания (возможен случайный выбор между данными вариантами оператора скрещивания).
3. Сформировать случайно начальную популяцию, состоящую из k особей $B_0 = \{A_1, A_2, \ldots, A_k\}$. При этом k определить эмпирически.
4. Вычислить приспособленность каждой особи $F_{Ai} = fit(A_i)$, $i=1 \ldots k$.
5. Выбрать двух особей A_c из популяции. $A_c = Get(B_t)$.
6. Произвести скрещивание (выбрав один из предложенных вариантов) и получить новую особь.
7. Произвести мутацию генов с некоторой вероятностью.

8. Оценить полученную особь функцией приспособленности и добавить в популяцию взамен худшего родителя (таким образом сохраняется количество особей популяции).

9. Выполнить пункты 5-8 m раз.

10. Увеличить счетчик эпох.

11. Проверить условие останова (предельное количество эпох или предельное количество особей одного генотипа в популяции).

РАЗРАБОТКА ПРИЛОЖЕНИЙ В СФЕРЕ МАШИННОГО ОБУЧЕНИЯ

Для реализации полноценного решения в сфере ПО необходимо учесть аппаратную и программную составляющую. Под аппаратной составляющей подразумевается следующее. Машинное обучение в ряде случаев требует подбор специального оборудования (или конфигурирование специальных серверов), рассчитанного на массивные вычислительные нагрузки или на тяжелые параллельные вычисления. Причем отличительной чертой можно назвать то, что мощное оборудование может потребоваться не только в производственной стадии проекта, но даже на ранних стадиях разработки для проведения экспериментов и соответствующего исследования моделей. Без мощного оборудования требуемые модели могут работать в 10-60 раз медленнее или просто не запуститься ввиду нехватки памяти (или видеопамяти).

В случае разработки встраеваемых решений может потребоваться сильная оптимизация кода и квалификация в написании кода под специальное оборудование (например, программируемые логические интегральные схемы). В случае разработки распределенных систем потребуется также применение специальных программных средств для организации распределенных вычислений.

Эти специализированные темы в данном пособии не рассматриваются.

Однако если речь не идет о разработке специализированного программно-аппаратного комплекса или встраевоемого решения, то наращивание вычислительных узлов на сегодняшний день является не столько инженерной проблемой сколько экономической.

Что же касается программной составляющей, то здесь стоит упомянуть следующее. На сегодняшний день огромное число различных алгоритмов уже реализовано в виде библиотек и пакетов, поэтому нет необходимости писать алгоритм нейронной сети или

агоритм решающих деревьев. За исключением опять же специализированных задач или исследовательских проектов.

Теоретически разработку систем МО можно вести на любом языке программирования, но при отсутствии специализированных библиотек сложность разработки может возрасти. Однако необходимо учесть, что специализированные средства создания моделей МО могут быть сложно применимы при реализации полноценных приложений. Поэтому нередко используют комбинацию технологий, когда предобработку данных и саму модель реализуют на одном языке программирования, а затем работающий прототип встраивают в приложение, написанное с использованием другой технологии. Одна из возможных архитектур реализации представлена на рисунке 43.

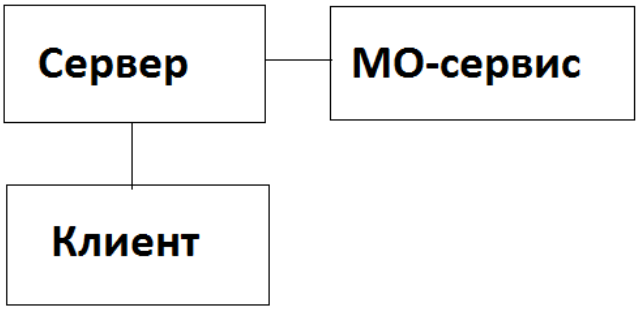


Рисунок 43. Возможная архитектура системы МО

Язык Python является одним из самых популярных языков для разработки в сфере машинного обучения и анализа данных ввиду очень простого синтаксиса, большого числа библиотек и развитого сообщества. Несмотря на то, что это интерпретируемый язык и выполнение кода на нем довольно медленное, большая часть библиотек реализована на быстрых языках типа Си, C++ и т. п. Поэтому на Python приходиться писать лишь высокоуровневое обращение к API библиотек. Что делает разработку на Python удобной, а решения – не уступающими по скорости со специальной реализацией на Си. В силу вышеизложенного будем работать именно с Python.

Однако прежде чем перейти непосредственно к Python, необходимо сказать, что при разработке систем МО, как правило, приходится пройти несколько типовых этапов. Таким образом, можно сформулировать общий алгоритм создания систем МО, который может быть изменен в силу особенностей конкретного проекта.

Алгоритм состоит из следующих шагов:

1. Сбор и предобработка данных;

2. Выбор метода решения, создание и настройка модели(подбор параметров, реализация прототипа);

3. Тестирование модели и совершенствование ее до достижения необходимой точности. При невозможности достижения точности – возврат к шагу 2;

4. Сохранение модели и интеграция ее в оболочку взаимодействия с пользователем;

5. Эксплуатация и сопровождение полученной системы и ее модернизация (при необходимости).

Основы работы с Python

Теперь перейдем к рассмотрению возможностей, предоставляемых Python для разработки систем МО. Общие сведения об этом языке можно посмотреть здесь: [30], а справочную информацию по синтаксису – здесь: [31]. В нашей же книге мы рассмотрим лишь то, что касается возможностей Python в плане создания приложений в сфере машинного обучения. Все примеры кода подготовлены на Python версии 2.7, но они могут быть адаптированы и под более старшие версии языка.

Как было сказано выше, сейчас Python является одним из наиболее распространенных языков программирования. Одним из его преимуществ является большое количество пакетов, решающих самые разные задачи. В данном пособии мы рекомендуем использовать библиотеки Pandas, NumPy и SciPy, которые существенно упрощают чтение, хранение и обработку данных. Вы также познако-

митесь с пакетом Scikit-Learn, в котором реализованы многие алгоритмы машинного обучения.

Необходимо отметить, что основными отличиями Python являются:

- Отсутствие завершающих символов в конце строки (точек, запятых и т. п.), что делает написание линейных конструкций очень «приятным» и быстрым занятием (а большинство программ для машинного обучения – это все-таки линейная математика, а не сложные многоуровневые системы).
- Нестрогая типизация. Не нужно объявлять тип данных при объявлении переменной, что опять же ускоряет процесс разработки модели.
- Язык Python интерпретируемый, кросс-платформенный и обладает хорошими средствами отладки.

Для разработки приложений нам нужно следующее:

- Язык, среда для разработки кода (IDE) и исполняемая среда. В нашем случае это Anaconda и PyCharm.
- Набор основных библиотек: Scikit-learn, Numpy, Pandas, Matplotlib, Theano, Keras.

Ниже представлены инструкции по установке и настройке компонент. Важный момент заключается в том, что все компоненты для работы в своем исходном состоянии «не родные» для систем Windows, поэтому в первую очередь эти инструкции относятся к пользователям Windows.

Инструкции по установке основных компонент:

1. Скачайте и установите AnacondaEnvironment (среда свободно распространяется на официальном сайте [32]). Anaconda необходима по большей части не как IDE, а как среда, в которой будет сразу установлено множество необходимых библиотек (развернутых соответствующим образом под Windows), таких как pip, numpy, matplotlib и еще пара десятков других. Также будет установлен и сам Python 2.7.

Обратите внимание!

- *Для корректной работы всех компонент аккаунт пользователя Windows, из под которого будет проводиться установка, должен содержать только латинские символы в своем имени.*

2. Рекомендуется установить PyCharm как основную IDE для разработки (хотя можно пользоваться самой SpiderAnaconda, которая будет установлена на предыдущем шаге) или JupyterNotebook. Однако если вы новичок в программировании или в Python, то рекомендуется именно PyCharm, так как эта среда сильно облегчает программирование и навигацию по коду. CommunityEdition распространяется бесплатно и скачать его можно здесь: [33].

3. Для проверки того, что все компоненты установлены корректно, запустите IDE, создайте новый файл (команда Alt+Insert или через меню File и пункт New) под именем testc кодом на Python, как показано на рисунке 44. Вставьте в файл код из листинга 1 и выполните его через пункт меню Run, как показано на рисунке 45.

Листинг 1

```
from sklearn import preprocessing
import numpy as np
X = np.array([[ 1., -1., 2.], [ 2., 0., 0.], [ 0., 1., -1.]])
X_scaled = preprocessing.scale(X)
print(X_scaled)
```

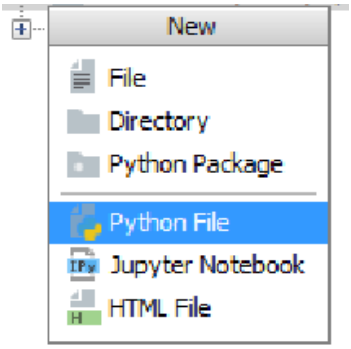


Рисунок 44. Новый файл Python

Обратите внимание!

- *Если в первом пункте меню Run нет имени вашего файла, то выберите третий пункт и в открывшемся окошке найдите имя вашего файла.*

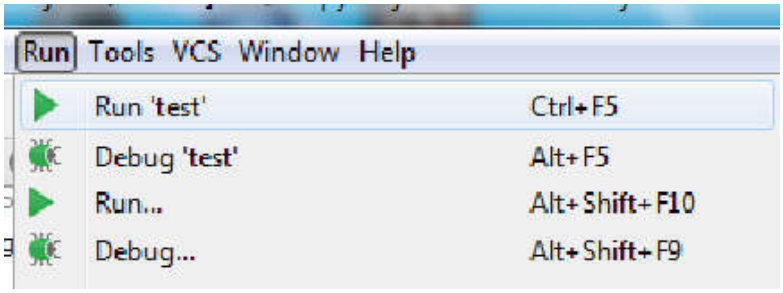


Рисунок 45. Запуск кода на выполнение

Если все установлено верно, то в консоли вы должны увидеть результат, представленный на рисунке 46.

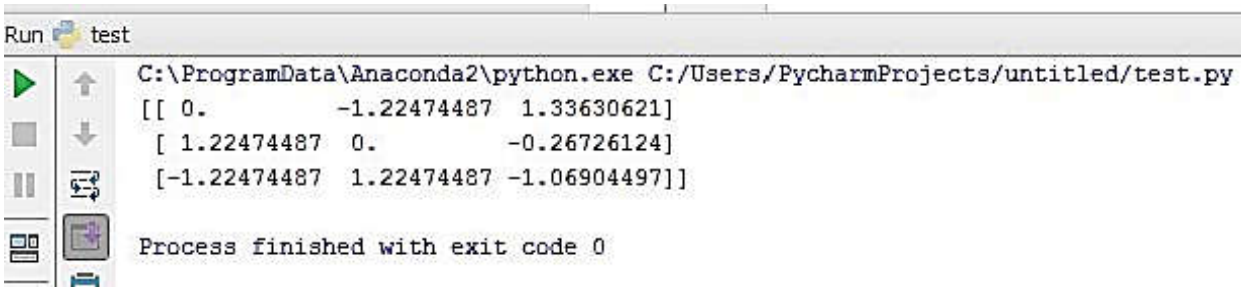


Рисунок 46. Корректная работа кода

Кроме описанного выше, нам понадобится установить и настроить еще ряд компонентов и библиотек. Инструкции по их установке следующие:

1. Скачайте и установите TDM-GCC (специальное средство компиляции для ОС Windows);

2. Откройте консоль Anacondaprompt или обычную командную строку Windows через команду cmd;

3. Выполните в консоли conda update conda;

4. Выполните в консоли conda update --all;

5. Выполните в консоли conda install mingw libpython;

6. Выполните в консоли conda install Theano;

7. Выполните в консоли pip install keras;

Обратите внимание!

- *Keras может быть настроен на tensorflowbackend, тогда надо внести изменения в файл C:/Users/AccauntName/.keras/keras.json и в строке backend=”tensorflow” написать theano;*

8. Запустите IDE, создайте новый файл с кодом на Python и выполните код из листинга 2 для проверки того, что все компоненты установлены корректно.

Листинг 2

```
import numpy as np
import sklearn.linear_model as lin
from keras.layers.core import Dense, Activation
from keras.models import Sequential
from keras.optimizers import RMSprop

x_train = np.array([[0.1, 0.3], [0.2, 0.2], [0.7, 0.8], [1.0, 0.9]])
y_train = np.array([0, 0, 1, 1]).reshape(4, 1)
x_test = np.array([[0, 0], [0.3, 0.3], [0.6, 0.7], [1, 1]])
model = Sequential()
model.add(Dense(2, init='lecun_uniform', input_shape=(2,)))
model.add(Activation('relu'))
model.add(Dense(1, init='lecun_uniform'))
model.add(Activation('linear'))
rms = RMSprop(lr=0.01)
model.compile(loss='mse', optimizer=rms)
epochs = 200
model.fit(x_train, y_train, batch_size=1, nb_epoch=epochs, verbose=0)
y_predict = model.predict(x_test, batch_size=1)
print(y_predict)
```

В результате в Output консоли не должно быть ошибок (код выхода равен 0), и должна будет распечататься информация, представленная в листинге 3.

Листинг 3

```
[[ 0.00339771]
[ 0.14396997]
[ 0.67507285]
[ 1.1803354 ]]
```

Мы установили основные, необходимые нам, компоненты. Теперь перейдем непосредственно к рассмотрению конкретных приемов работы с ними.

В данном пособии будет приведено описание только тех операторов и функций, которые непосредственно используются при решении конкретной задачи. Для получения прочей информации по Python рекомендуется воспользоваться справочником или же поисковой системой Google.

Обратите внимание!

- *Если у вас есть конкретный вопрос, например, о том, как выбрать последний элемент в одномерном или двумерном массиве, или о том, как отсортировать массив и т. п., то лучшее решение – написать этот вопрос в Google на английском;*
- *Если ваш уровень английского недостаточен, то просто сформулируйте вопрос на русском максимально коротко, затем вставьте в GoogleTranslate и затем – в поисковик. С вероятностью 90% на такие конкретные вопросы вы найдете очень конкретные примеры кода на сайте StackOverflow.*

Элементарные операции с данными

Рассмотрим работу по предобработке данных и поиску простых закономерностей в них средствами Python, а именно средствами пакетов Numpy и Pandas. Более подробно с их функциями можно ознакомиться, например, здесь: [34].

Для импорта модуля Numpy необходимо написать следующую строку кода (листинг 4).

Листинг 4

```
Import numpy as np
```

Обратите внимание!

- *Numpy импортируется с псевдонимом np, через который в дальнейшем будет обращение к модулю.*

Основными единицами данных в машинном обучении являются векторы и матрицы. С точки зрения программирования такие структуры представляют собой одномерные и двумерные массивы или специальные объекты-таблицы.

Обратите внимание!

- *Если в тексте написано просто – вектор или матрица, то значит идет речь о одномерном или двумерном массиве Numpy. В других случаях будет специально написано в каком формате представлены данные (например, DataFrame, о чем речь пойдет ниже).*

Для того чтобы рассмотреть способы работы с данными, нам необходимо получить данные для работы. Так как модели, разрабатываемые на Python, в 99% случаев используют полученные откуда-то данные или же генерируют тестовые, то мы опишем три способа получения данных: прямое объявление, случайная генерация и загрузка из csv-формата.

Начнем с первого. Объявить матрицу можно, используя код из листинга 5. Как видите, ничего сложного здесь нет: ни предварительного объявления данных, ни выделения памяти, ни объявления типа данных.

Листинг 5

```
Z = np.array([[4, 5, 0],
              [9, 9, 9]])
```

Теперь рассмотрим второй способ: сгенерируем случайную матрицу, состоящую из 6 строк и 5 столбцов. Элементы ее будут являться случайными числами из нормального распределения. Функция для генерации таких чисел: np.random.normal.

Ее параметры:

- loc: среднее нормального распределения (в нашем случае 1);
- scale: стандартное отклонение нормального распределения (его значение в нашем случае равно 10);
- size: размер матрицы (в нашем случае (6, 5)).

Код генерации матрицы X и ее распечатки представлены в листинге 6.

Листинг 6

```
X = np.random.normal(loc=1, scale=10, size=(6, 5))
print X
```

Далее рассмотрим способ загрузки данных из файла в csv-формате, который предназначен для хранения табличных данных. Как правило, в файлах этого формата столбцы разделяются запятой, а первая строка содержит их имена.

Допустим, у нас есть файл «titanic.csv», содержащий данные в виде, представленном на рисунке 47.

Обратите внимание!

- *Небольшие файлы csv (до 1 Гб) можно достаточно удобно посмотреть в Excel. Чтобы корректно загрузить CSV файл, нужно сначала открыть Excel, создать пустую книгу, а затем вызвать соответствующего мастера загрузки через вкладку «Данные»=>«Из текста»..*

A	B	C	D	E	F	G	H	I	J	K	L
PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, M	male	22	1	0	A/5 21171	июл.25		S
2	1	1	Cumings, I	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen	female	26	0	0	STON/O2. 3101282	7.925		S
4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S

Рисунок 47. Файл с данными

Для загрузки этих данных нам необходимы средства библиотеки pandas. Код ее импорта, а также код загрузки и распечатки загруженных данных представлены в листинге 7. Там же представлен код записи данных в csv-файл.

Листинг 7

```
# -*- coding: utf-8 -*-
import csv
import pandas
#чтение из файла
data = pandas.read_csv('titanic.csv', index_col='PassengerId')
print data
#запись в файл данных в исходном виде
data.to_csv('q.csv')
#запись в файл данных как массива значений
with open('test.csv', 'w') as csvfile:
  spamwriter = csv.writer(csvfile, delimiter=';',quotechar='|', quoting=csv.QUOTE_MINIMAL)
  spamwriter.writerow (data.as_matrix())
```

Обратите внимание!

- *Символ # - знак комментария.*
- *Для того чтобы закомментировать (или раскомментировать) несколько строк, необходимо выделить их и нажать сочетание клавиш Ctrl+правый слеш.*
- *Для того чтобы писать комментарии в Python на русском языке, необходимо добавить следующую строку в начало файла с кодом:# -*- coding: utf-8 -*-*

При выполнении кода из листинга 7 данные будут загружены в виде DataFrame, с помощью которого можно удобно работать с ними. Параметр index_col='PassengerId' означает, что колонка PassengerId задает нумерацию строк данного DataFrame.

DataFrame – первичная структура данных pandas, представляющая собой двумерную, изменяемую по размеру, потенциально гетерогенную структуру табличных данных с маркированными осями (строками и столбцами).

В виде DataFrame над данными удобно производить различные манипуляции сортировки, выборки, применения функций построчно или по колонкам и т. п. Однако DataFrame – это комплексный объект высокого уровня и его нельзя использовать напрямую для обучения моделей. Перед этим его нужно перевести в обычную цифровую матрицу, например, следующим образом (листинг 8), и дальше работать с ним как с обычным двумерным массивом данных.

Листинг 8

```
x_test= data[['ColumnName_1', 'ColumnName_2']].as_matrix()
```

Вернемся к листингу 7. В нем представлены два способа записи в файл. Первый – через метод самого DataFrame, второй – через метод библиотеки csv. Необходимо отметить, что второй способ может быть использован не только для работы с DataFrame, но и с любыми данными, которые надо записать в csv-файл.

Мы рассмотрели основные способы получения данных, теперь перейдем к работе с ними и продемонстрируем некоторые возможности библиотеки Numpy по операциям с матрицами на конкретных задачах из практики.

Допустим, перед нами стоит следующая задача: есть данные о ежедневной выручке по филиалам компании за первые 12 дней месяца, записанные в виде таблицы (рисунок 48) и сохраненные в файле «data.csv». Выведем номер филиала, преодолевшего по прибыли порог, равный 1000.

Первое, что нам необходимо сделать, – загрузить данные. Однако, чтобы pandas корректно обработал файл, нам нужно убрать из него кириллицу и убедиться, что разделители в csv действительно запятые. Для этого мы можем поместить файл в папку проекта pyCharm, двойным щелчком мыши на нем (в окне проекта) открыть его и привести к виду, показанному на рисунке 49. Теперь для чтения данных, преобразованию их в матрицу и распечатки полученного массива мы можем использовать код из листинга 9.

A	B	C	D
День месяца	Филиал1	Филиал2	Филиал3
1	20	30	40
2	13	35	35
3	14	453	6
4	53	13	57
5	47	13	577
6	53	13	75
7	35	13	57
8	23	13	46
9	53	31	46
10	65	123	78
11	75	24	68
12	57	42	86

Рисунок 48. Файл с данными для задачи

test.py × data.csv ×

```
1  Day,Fil1,Fil2,Fil3
2  1,20,30,40
3  2,13,35,35
4  3,14,453,6
5  4,53,13,57
6  5,47,13,577
7  6,53,13,75
8  7,35,13,57
9  8,23,13,46
10 9,53,31,46
11 10,65,123,78
12 11,75,24,68
13 12,57,42,86
```

Рисунок 49. Корректный файл с данными

Обратите внимание!

- *Если исходный файл создан через Excel, то разделителями могут быть точки с запятой. Заменить их на запятые можно при редактировании файла в PyCharm через цепочку пунктов меню «Edit»=>«Find» =>«Replace».*

Листинг 9

```
import pandas
data= pandas.read_csv('data.csv', index_col='Day')
x_test= data[['Fil1', 'Fil2', 'Fil3']].as_matrix()
print x_test
```

Если все сделано верно, в консоли должен появиться результат, показанный на рисунке 50.

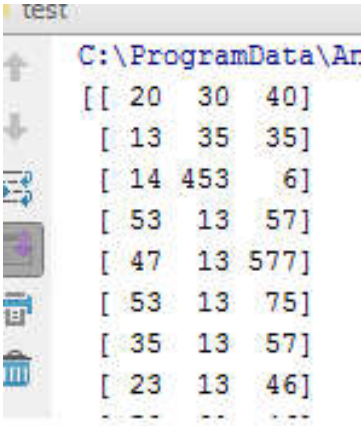


Рисунок 50. Загруженные данные

Обратите внимание!

- *При преобразовании к матрице мы указываем в кавычках имена столбцов с данными по филиалам и не указываем столбец Day, так как содержащиеся в нем данные для задачи не нужны.*

Теперь перейдем непосредственно к решению поставленной задачи. С точки зрения работы с матрицей, для получения ответа нам

необходимо посчитать сумму по каждому столбцу и вывести номера тех, чья сумма превысит 1000.

Функция для подсчета суммы: np.sum. В качестве параметров она принимает матрицу, для которой необходимо посчитать сумму и измерение (строки или столбцы), которое необходимо суммировать. Измерение (axis) задается цифрой (для двумерной матрицы 0 – строки, 1 – столбцы).Если этот параметр не задать, то результат функции будет рассчитан для всей матрицы целиком. Результатом выполнения операции будет массив с соответствующими суммами. Библиотека Numpy предоставляет возможности применения к матрицам (и массивам) логических операций, причем применяемых поэлементно. Соответственно, результатом такой операции будет матрица такого же размера, в ячейках которой будет записано либо True, либо False (удовлетворяет текущий элемент условию или нет). Индексы элементов со значением True можно получить с помощью функции np.nonzero. Функция в качестве параметра принимает матрицу, в которой необходимо отыскать ненулевые элементы. Заметим, что в нашем случае мы можем сразу передать логическое выражение, составленное из массива сумм и самого условия, тогда элементы со значением False будут интерпретироваться как нулевые. Код решения представлен в листинге 10. Результат работы кода – на рисунке 51. Ответ к задаче: третий филиал.

Листинг 10

```
import numpy as np
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
x_test= data[['Fil1', 'Fil2', 'Fil3']].as_matrix()
#print x_test
r = np.sum(x_test, axis=0)
print (r)
print np.nonzero(r> 1000)
```

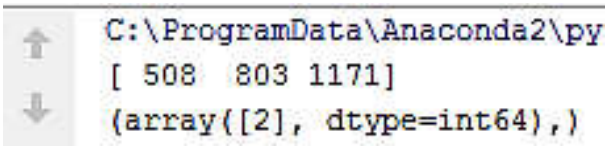


Рисунок 51. Решение задачи

Теперь решим следующую задачу: определим, в какой день суммарная выручка по филиалам превысила 500. Код решения представлен в листинге 11, а результат – на рисунке 52.

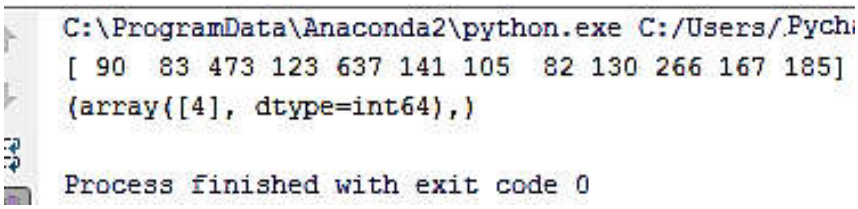


Рисунок 52. Решение задачи

Листинг 11

```
import numpy as np
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
x_test= data[['Fil1', 'Fil2', 'Fil3']].as_matrix()
r = np.sum(x_test, axis=1)
print (r)
print np.nonzero(r>500)
```

Код нахождения филиала с максимальной средней выручкой представлен в листинге 12, а результат показан на рисунке 53.

Листинг 12

```
import numpy as np
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
x_test= data[['Fil1', 'Fil2', 'Fil3']].as_matrix()
r = np.mean(x_test, axis=0)
print (r)
print np.nonzero(r== np.max(np.mean(x_test, axis=0)))
```

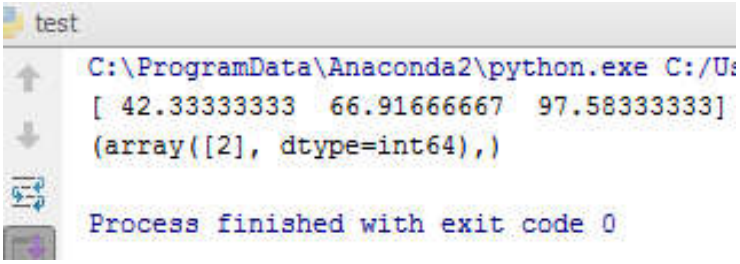


Рисунок 53. Номер филиала с максимальной средней выручкой

Номер филиала с максимальной величиной стандартного отклонения прибыли можно получить, используя код из листинга 13.

Далее нам необходимо более подробно рассмотреть работу со структурой DataFrame, но перед этим упомянем еще две полезные функции Numpy, которые могут понадобиться в последующем: функцию генерации единичной матрицы (np.eve) и функцию вертикальной стыковки матриц (np.vstack). Первая в качестве параметра принимает количество строк (оно же количество столбцов), вторая – матрицы, которые нужно объединить. В листинге 14 показан пример генерации и объединения двух единичных матриц.

Листинг 13

```
import numpy as np
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
x_test= data[['Fil1', 'Fil2', 'Fil3']].as_matrix()
r = np.std(x_test, axis=0)
print (r)
print np.nonzero(r== np.max(np.std(x_test, axis=0)))
```

Листинг 14

```
import numpy as np
A = np.eye(3)
B = np.eye(3)
print A
print B
AB = np.vstack((A, B))
```

Работа с DataFrame

Мы рассмотрели основные моменты работы с матрицами, поэтому теперь вернемся к другой структуре данных: DataFrame. Как мы помним, она является таблицей, где колонки имеют заголовки, а строки, кроме номеров, могут иметь дополнительные индексы в виде цифр или имен (по сути, индексы представляют собой отдельную, специальную колонку в таблице DataFrame, которая не относится к данным). Для того чтобы посмотреть, что представляют собой данные, можно воспользоваться несколькими способами.

Если мы хотим распечатать первые 5 строк данных, то можем воспользоваться кодом из листинга 15, в случае, если указан только один индекс, или же воспользоваться методом head() DataFrame, как показано в листинге 16.

Листинг 15

```
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
print data[:5]
```

Листинг 16

```
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
print data.head()
```

Если же нас интересует содержимое конкретного столбца, то можно использовать квадратные скобки и название столбца, как показано в листинге 17.

Листинг 17

```
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
print data['Fil1']
```

Обратите внимание!

- *Код из листинга 16 не применим к индексному столбцу.*

Для подсчета некоторых статистик (количества, среднее, максимум, минимум) можно также использовать методы объекта DataFrame. В листинге 18 приведен пример кода, считающего количество повторов каждого из значений в столбце с именем 'Fil1'. Результат работы кода показан на рисунке 54. Значения суммы, минимума, максимума, среднего, количества и среднеквадратического отклонения вычисляются аналогично с использованием соответствующих функций (sum, min, max, mean, count, std).

Листинг 18

```
import pandas
data = pandas.read_csv('data.csv', index_col='Day')
print data['Fil1'].value_counts()
```

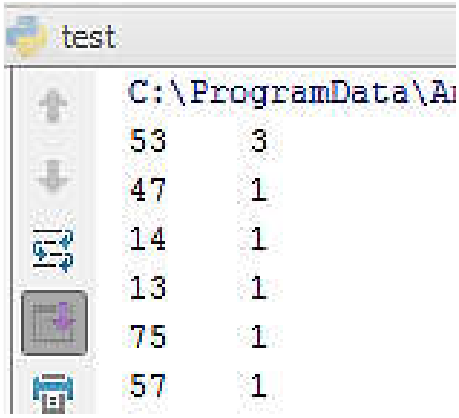


Рисунок 54. Результат работы кода из листинга 18

Теперь рассмотрим еще несколько более сложных функций работы с DataFrame на примере некоторой таблицы, представленной на рисунке 55.

В листинге 19 представлены интересующие нас фрагменты кода с комментариями. Как можно заметить, возможности DataFrame весьма обширны, поэтому здесь приведены лишь основные моменты, а с остальным вам предлагается ознакомиться самостоятельно.

	first_name	company_name	address	city	county	postal	phone1	phone2
last_name								
Villamarin	Antonio	Combs Sheetmetal	353 Standish St #8264	Little Parndon and Hare Street	Hertfordshire	CM20 2HT	01559-403415	01388-777812
Glasford	Antonio	Saint Thomas Creations	425 Howley St	Gaer Community	Newport	NP20 3DE	01463-409090	01242-318420
Heilig	Antonio	Radisson Suite Hotel	35 Elton St #3	Ipplepen	Devon	TQ12 5LL	01324-171614	01442-946357

Рисунок 55. Структура таблицы с данными

Обратите внимание!

- *Для лучшего понимания кода листинга 19 вам рекомендуется почитать про Лямбда функции в Python, например, здесь: [35];*
- *Более подробно со списком методов DataFrame можно познакомиться в документации: [36];*
- *Ссылка на уроки по Pandas: [37].*

Листинг 19

```
#-*- coding: utf-8 -*-
# Выбор элементов DataFrame с использованием iloc
# Строки:
data.iloc[0] # выбирает первую строку таблицы.
data.iloc[1] # выбирает вторую строку таблицы
data.iloc[-1] # выбирает последнюю строку таблицы

# Столбцы:
data.iloc[:,0] # выбирает первую колонку таблицы
data.iloc[:,1] # выбирает вторую колонку таблицы
data.iloc[:,-1] # выбирает последнюю колонку таблицы

# Множественный выбор
data.iloc[0:5] # выбирает первые пять строк таблицы
data.iloc[:, 0:2] # выбирает первые две колонки таблицы со всеми строками

# Выбирает строки с индексами 'Andrade' и 'Veness' (в случае текстовых индексов),
#со всеми колонками между колонок с именем 'city' и 'email'
data.loc[['Andrade', 'Veness'], 'city':'email']
```

Окончание листинга 19

```
# Выбирает те же строки, только с колонками 'first_name', 'address' и 'city'
data.loc['Andrade':'Veness', ['first_name', 'address', 'city']]

# Выбирает строки с firstname равным Antonio и выбирает только колонки между
#колонками 'city' и 'email'
data.loc[data['first_name'] == 'Antonio', 'city':'email']

# Выбирает строки со всеми колонками, где в колонке email встречаются ячейки,
#заканчивающиеся на 'hotmail.com'
data.loc[data['email'].str.endswith("hotmail.com")]

#Выбирает строки, где first_name равно одному из следующих значений
data.loc[data['first_name'].isin(['France', 'Tyisha', 'Eric'])]

# Выбирает строки, где в колонке first_name встречается Antonio и в колонке email
#встречается gmail.com
data.loc[data['email'].str.endswith("gmail.com") & (data['first_name'] == 'Antonio')]

# Выбирает строки, у которых в колонке id есть значения от 100 до 200, и возвращает
#только колонки 'postal' и 'web'
data.loc[(data['id'] > 100) & (data['id'] <= 200), ['postal', 'web']]

#Лямбда функция, которая возвращает True/False переменную.
# Выбирает строки, где ячейка в колонке company_name имеет 4 слова.
data.loc[data['company_name'].apply(lambda x: len(x.split(' ')) == 4)]

#Распечатка заголовков
print list(data)
```

Предобработка данных. Стандартизация и нормализация

Мы рассмотрели основные моменты работы с данными, теперь перейдем к такому вопросу, как предобработка данных. Прежде чем обучать и использовать большинство моделей, необходимо привести исходные данные к единообразному виду (к единому диапазону). Единый диапазон определяется через математическое ожидание выборки и разброс (дисперсию). То есть нужно, например, чтобы все данные были в диапазоне от 0 до 1 или от –1 до 1. Как было сказано

в одном из предыдущих разделов, достигается такое приведение посредством двух процессов\методов: стандартизации данных и нормализации.

Можно провести стандартизацию самостоятельно, используя Numpy. Для этого вычтите из каждого столбца его среднее значение, а затем поделите на его стандартное отклонение. Или же можно использовать библиотеку scikit-learn, как показано в листинге 20.

Однако, чаще всего, кроме нормировки одной матрицы, потребуется нормировать несколько матриц по одним и тем же шкалам (для каждого столбца\признака будут свои коэффициенты смещения и масштабирования).

Так если тестовая\рабочая выборки поступают в систему онлайн, то их потребуется нормировать по тем же коэффициентам, что и тренировочную (рассчитанным на тренировочной выборке). Иначе нормировка будет некорректной.

Листинг 20

```
from sklearn import preprocessing
import numpy as np
X = np.random.normal(loc=1, scale=10, size=(6, 5))
X_scaled = preprocessing.scale(X)
print X
print X_scaled
```

Вместо того чтобы хранить матрицы коэффициентов нормировки, гораздо удобнее воспользоваться классом StandardScaler из scikit-learn. Сперва нужно «обучить» его на тренировочных данных (методом fit), а потом преобразовывать разные матрицы одинаковым образом. Работа с этим классом показана в листинге 21.

Листинг 21

```
from sklearn import preprocessing
import numpy as np
X = np.random.normal(loc=1, scale=10, size=(6, 5))
X_test = np.random.normal(loc=1, scale=10, size=(6, 5))
```

Окончание листинга 21

```
scaler = preprocessing.StandardScaler().fit(X)
train_data_scaled = scaler.transform(X)
test_data_scaled = scaler.transform(X_test)
print train_data_scaled
print test_data_scaled
```

Как было сказано в одном из предыдущих разделов, стандартизация смещает значения векторов (выравнивает относительного единого центра в нуле), а также производит выравнивание разброса. Однако значения разных векторов не будут в одинаковом диапазоне (от –1 до 1, например). Они будут лишь иметь стандартный разброс в рамках вектора\столбца\признака. Для того чтобы достичь одинакового масштаба всех векторов, необходима нормализация.

Средствами scikit-learn нормализацию можно выполнить следующим образом (листинг 22).

Листинг 22

```
#-*- coding: utf-8 -*-
from sklearn import preprocessing
import numpy as np
X=[[1., -1.,2.],
  [2., 0., 0.],
  [0., 1., -1.]]

#нормализация max-нормой
X_normalized_max = preprocessing.normalize(X, norm='max', axis=0)

#нормализация нормой l1
X_normalized_max = preprocessing.normalize(X, norm='l1', axis=0)

#нормализация нормой l2
X_normalized_max = preprocessing.normalize(X, norm='l2', axis=0)
```

Более подробную информацию о методах библиотеки scikit-learn по предобработке данных можно найти здесь: [38].

Работа с деревьями решений

Мы рассмотрели методы предобработки данных. Теперь перейдем к следующему – к деревьям решений. Как было сказано в одном из предыдущих разделов, решающие деревья относятся к классу логических методов. Их основная идея состоит в объединении определенного количества простых решающих правил, благодаря чему итоговый алгоритм является интерпретируемым. Как следует из названия, решающее дерево представляет собой бинарное дерево, в котором каждой вершине сопоставлено некоторое правило вида «j-й признак имеет значение меньше b». В листьях этого дерева записаны числа-предсказания. Чтобы получить ответ, нужно стартовать из корня и делать переходы либо в левое, либо в правое поддерево в зависимости от того, выполняется правило из текущей вершины или не выполняется.

Одна из особенностей решающих деревьев заключается в том, что они позволяют получать важности всех используемых признаков. Важность признака можно оценить на основе того, как сильно улучшился критерий качества благодаря использованию этого признака в вершинах дерева.

Рассмотрим данные о пассажирах «Титаника» (их можно скачать здесь: [39]). Решим на них задачу классификации, в которой по различным характеристикам пассажиров требуется найти у выживших пассажиров два наиболее важных признака (из четырех рассматриваемых: пол, класс, возраст, цена билета).

В библиотеке scikit-learn решающие деревья реализованы в классах sklearn.tree.DecisionTreeClassifier (для классификации) и sklearn.tree.DecisionTreeRegressor (для регрессии). Обучение модели производится с помощью функции fit. Найти важность признаков можно, имея уже обученный классификатор: его поле feature_importances_ содержит массив «важностей» признаков. Индекс в этом массиве соответствует индексу признака в данных. Стоит обратить внимание, что данные могут содержать пропуски.

Pandas выгружает такие значения как nan (not a number). Для того чтобы проверить, является ли число nan'ом, можно воспользоваться функцией numpy isnan.

Перейдем к решению задачи. Первое, что нам необходимо сделать, – внимательно посмотреть на данные перед загрузкой. Ранее мы уже предположили, что среди них могут быть пропуски, и определились с методом решения этой проблемы, но при изучении информации мы должны увидеть еще одну: признак Sex имеет строковые значения, тогда как все остальные – числовые. Следовательно, нам необходимо привести и его к числовому виду. Например, заменим male на 0, остальное – на 1.

Для этого определим функцию, которую применим к нашему массиву данных. В Python функции начинаются с ключевого слова def, и их операторы (тело функции) обязательно имеют отступ от начала строки. Применение функции к объекту выполняется оператором apply. Код функции представлен в листинге 23.

Листинг 23

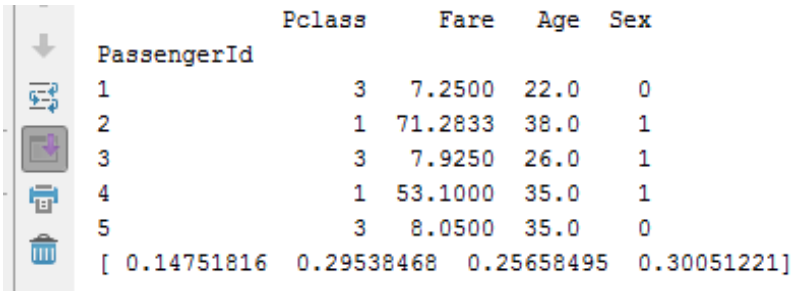
```
def Sex_to_bool(sex):
    if sex == "male":
        return 0
    return 1
```

Алгоритм решения задачи будет следующим:

1. Загрузить выборку из файла titanic.csv с помощью пакета Pandas.
2. Оставить в выборке четыре признака: класс пассажира (Pclass), цену билета (Fare), возраст пассажира (Age) и его пол (Sex). Привести пол к числовому виду и убрать из выборки пустые значения.
3. Выделить целевую переменную (она записана в столбце Survived).

4. Обучить решающее дерево с параметром random_state=241 и остальными параметрами по умолчанию (речь идет о параметрах конструктора DecisionTreeClassifier).
5. Вывести важности признаков.

Код решения задачи (с комментариями) – в листинге 24. Результат работы – на рисунке 56.



```
             Pclass     Fare   Age  Sex
PassengerId
1                 3   7.2500  22.0    0
2                 1  71.2833  38.0    1
3                 3   7.9250  26.0    1
4                 1  53.1000  35.0    1
5                 3   8.0500  35.0    0
[ 0.14751816  0.29538468  0.25658495  0.30051221]
```

Рисунок 56. Важности признаков

Листинг 24

```
-*- coding: utf-8 -*-
import pandas
from sklearn.tree import DecisionTreeClassifier
import numpy as np

data = pandas.read_csv('titanic.csv', index_col='PassengerId')
#функция для приведения пола к числу
def Sex_to_bool(sex):
    if sex == "male":
        return 0
    return 1
#приведение пола к числу
data['Sex'] = data['Sex'].apply(Sex_to_bool)
#выгрузка непустых данных
data=data.loc[(np.isnan(data['Pclass'])==False) & (np.isnan(data['Fare'])==False)
&(np.isnan(data['Age'])==False) & (np.isnan(data['Sex'])==False)&
(np.isnan(data['Survived'])==False)]
#отбор нужных столбцов
corr = data[['Pclass', 'Fare', 'Age', 'Sex']]
#респечатка первых 5 строк данных
print corr.head()
```

Окончание листинга 24

```
#определение целевой переменной
y = data['Survived']
#создание и обучение дерева решений
clf = DecisionTreeClassifier(random_state=241)
clf.fit(corr, y)
#получение и распечатка важностей признаков
importances=clf.feature_importances_
print importances
```

Таким образом мы видим, что наиболее важными признаками будут пол и цена билета. Однако по описанному решению стоит сделать одно важное замечание. При преобразовании пола к числовому виду мы исказили характер данных: ранее объекты столбца «пол» не могли быть математически сравнимы между собой, а после наших преобразований эта характеристика у них появилась. Это привело к неоднозначности итогового решения: при male=0 массив ответа имеет вид: [0.14751816 0.29538468 0.25658495 0.30051221], а при female=0 – [0.14000522 0.30343647 0.2560461 0.30051221]. Конечно, в нашей ситуации изменение незначительное и не влияет на итоговый ответ, но в другой ситуации это может быть не так. Поэтому вам предлагается самостоятельно подумать, как избежать подобного искажения данных.

Работа с линейной регрессией

Мы рассмотрели работу с решающими деревьями, теперь же перейдем к рассмотрению регрессии. Начнем с линейных моделей, приведя в качестве примера решение задачи прогнозирования уровня зарплаты по данным вакансий.

Линейные методы хорошо подходят для работы с разреженными данными. К таковым относятся, например, тексты. Это можно объяснить высокой скоростью обучения и небольшим количеством параметров, благодаря чему удается избежать переобучения.

Линейная регрессия имеет несколько разновидностей в зависимости от того, какой регуляризатор используется. В рамках предлагаемой задачи мы будем использовать гребневую регрессию, где применяется квадратичный, или L2-регуляризатор. Эта модель реализована в классе sklearn.linear_model.Ridge. Более подробно о реализации гребневой регрессии можно почитать здесь: [40].

Исходные данные объемом 60 000 записей хранятся в файле «salary_train.csv», тестовые данные (2 записи) хранятся в файле «salary_test_mini.csv» (скачать их можно отсюда: [41]). Оба файла имеют вид, представленный на рисунке 57.

В данном случае столбцы A-C будут содержать значения входных признаков, а последний столбец – целевой переменной.

	A	B	C	D
1	FullDescription	LocationNormalized	ContractTime	SalaryNormalized
2	International Sales Manager London ****k ****k Uncapped Commission Digital Marketing/I	London	permanent	33000
3	An ideal opportunity for an individual that has gained experience in the field of M A to furth	London	permanent	50000
4	Online Content and Brand Manager// Luxury Retail // Up to ****k InstinctвЂ™s Central Lond	South East London	permanent	40000
5	A great local marketleader is seeking a permanent Payroll and Purchase Ledger Assistant to j	Dereham	permanent	22500
6	Registered Nurse / RGN Nursing Home for Young Adults Location; Sutton Coldfield Part Time	Sutton Coldfield		20355
7	Sales and Marketing Assistant will provide administrative support to the staff of the Sales/S	Crawley		22500
8	Vacancy Ladieswear fashion Area Manager / Regional Manager Northern Ireland where will	UK	permanent	32000
9	Reference: LR/JAN/**** Our client is one of the largest Multi Disciplinary Engineering Consu	Bristol	permanent	30000
10	Sponsorship Manager London The Company A market leading event and conference busines	Central London	permanent	31500
11	About Barclays Barclays moves, lends, invests and protects money for customers and clients	South East London	permanent	42499
12	Counter Sales Person c****k dependent on experience Banchory Temporary MADE TO MEAS	Banchory	contract	16000
13	Central London Competitive Salary Bonus & Benefits Summary Aegis Media Limited (AML) i	Gipsy Hill	permanent	47500
14	We are recruiting for a dynamic organisation based near pontefract in Wakefield. They requ	Wakefield	permanent	20000
15	Composite Design Engineer required for a permanent role for Oxfordshire. Need relevant d	Abingdon	permanent	34000
16	Working managing a team to maintain the Home by cleaning, dusting, vacuuming and polish	Northampton		15360
17	HCL Nursing are actively recruiting skilled registered general nurses to work in various hospi	Bradford	permanent	39811
18	HGV / Commercial vehicle Customer service advisor My client based in North London is curr	Northamptonshire	contract	23000
19	Break into Banking .NET Developer (C/Agile/WinForms/WebForms/SQL)********K This is a	London	permanent	47500
20	Registered Nurse (RGN) is required to work within the elderly nursing home environment. T	Tiverton		24500
21	Can you drive effective analytical solutions for the European Corporate compensation and b	Uxbridge		32000
22	nbsp; My client is an small, but high profile international charity who are looking to looking t	London	permanent	45000
23	Specialist Clinical Negligence Practice based in Manchester City Centre has a new opportuni	Lancashire	permanent	20000
24	Job description: вЂў Answer contacts promptly and professionally вЂў Log/Validate all conta	UK	permanent	21000
25	YEAR **** TEACHER ROLE Servoca Education is currently looking for a Year **** class teacher	Surrey	contract	34800

Рисунок 57. Исходные данные

Как можно заметить, первый столбец содержит объемный текст, который необходимо предобработать для загрузки в модель. Например, извлечь TF-IDF-признаки, воспользовавшись классом sklearn.feature_extraction.text.TfidfVectorizer, который преобразует массив текста в матрицу, содержащую TF-IDF-признаки. Более

подробно об этом классе можно почитать здесь: [42]. Использование его для решаемой задачи показан в листинге 25.

Листинг 25

```
vectorizer = TfidfVectorizer(min_df=10)
train_text_feature_matrix = vectorizer.fit_transform(data_train['FullDescription'])
idf = vectorizer.idf_
print dict(zip(vectorizer.get_feature_names(), idf))
```

В данном случае поле idf_ класса TfidfVectorizer содержит вектор с глобальными весовыми коэффициентами. Строка кода print dict(zip(vectorizer.get_feature_names(), idf)) выведет в консоль массив данных вида: {u'pre': 5.6339293213815242, u'paperless': 9.0576775285655771, u'peoplewithchemistry': 9.6042212349336467, u'rfps': 9.1117447498358519, u'yellow': 8.957594070008593...}.

Теперь, что касается оставшихся столбцов-признаков: LocationNormalized и ContractTime являются строковыми, и поэтому с ними нельзя работать напрямую. Применим к ним one-hot-кодирование. Для рассматриваемого нами языка Python оно реализовано в классе sklearn.feature_extraction.DictVectorizer, который преобразует списки сопоставленных пар признак-значение в массивы, с которыми может работать Numpy, или в разреженные матрицы (scipy.sparse-matrices) для использования с классами-«измерителями качества» scikit-learn (scikit-learnestimators). Когда значения признаков строковые, этот трансформер выполняет бинарное one-hot-кодирование: для каждого из возможных строковых значений признака строится признак с логическим значением, говорящий, принимает данный признак указанное строковое значение или нет. Например, пусть на входе есть признак «c», который может принимать значения «кот» и «дог». На выходе будет два признака: один – «c = кот», другой – «c = дог». Признаки, которые не представлены в обучающей выборке, будут иметь нулевое значение в результирующей матрице или результирующем массиве.

Подробнее о DictVectorizer можно прочитать здесь: [43]. Пример использования этого класса для указанных данных приведен в листинге 26.

Листинг 26

```
# -*- coding: utf-8 -*-
from sklearn.feature_extraction import DictVectorizer
.....
enc = DictVectorizer()
train_dic = data_train[['LocationNormalized', 'ContractTime']].to_dict('records')
test_dic = data_test[['LocationNormalized', 'ContractTime']].to_dict('records')
X_train_categ = enc.fit_transform(train_dic)
X_test_categ = enc.transform(test_dic)
```

При этом считается, что в data_train загружены данные обучающей выборки, а в data_test – данные, на которых будет проверяться работа модели.

В результате работы кода из листинга 26 train_dic (также как и test_dic) будет содержать множество записей, таких как: {['LocationNormalized': 'UK', 'ContractTime': 'permanent'], ['LocationNormalized': 'Bradford', 'ContractTime': 'permanent']}. X_train_categ будет содержать данные, представленные на рисунке 58.

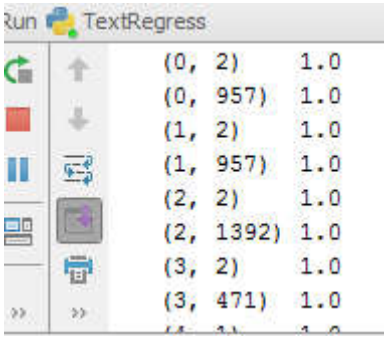


Рисунок 58. Преобразованные данные

Интерпретировать их можно следующим образом. X_train_categ – это по сути матрица, но только не в классическом формате. Если точнее, то X_train_categ – это разреженная матрица, которая в сокращенном виде описывает некую полную матрицу (назовем ее X_Full).

В X_train_categ указано, в каких позициях X_Full стоят единицы. Так (0, 2) 1.0 – означает, что в нулевой строке и втором столбце стоит единица. Во всех остальных ячейках матрицы X_Full, которые не обозначены в X_train_categ, стоят нули. X_Full – это матрица размера M на N, где M – число строк, которое определяется количеством объектов в выборке, а N – число уникальных слов в столбцах ['LocationNormalized', 'ContractTime'] исходной таблицы.

Теперь вернемся к предобработке данных. Так как в строковых данных есть пропуски, то нам потребуется заменять их на специальные строковые величины (например, 'nan'). Для этого можно использовать код из листинга 27.

Листинг 27

```
data_train['LocationNormalized'].fillna('nan', inplace=True)
```

С учетом изложенного выше, алгоритм решения поставленной задачи будет следующим:

1. Загрузить данные об описаниях вакансий и соответствующих годовых зарплатах из файла salary-train.csv.
2. Провести его предобработку:
   2.1. Привести тексты к нижнему регистру (text.lower()).
   2.2. Заменить все, кроме букв и цифр, на пробелы для облегчения дальнейшего разделение текста на слова. Для такой замены в строке text подходит следующий вызов: re.sub('[^a-zA-Z0-9]', ' ', text). Также можно воспользоваться методом replace у DataFrame, чтобы сразу преобразовать все тексты, например, так: train['FullDescription'] = train['FullDescription'].replace('[^a-zA-Z0-9]', ' ', regex = True)
   2.3. Применить TfidfVectorizer для преобразования текстов в векторы признаков. Оставить только те слова, которые встречаются хотя бы в 5 объектах. Для этого использовать параметр min_df у TfidfVectorizer.

2.4. Заменить пропуски в столбцах LocationNormalized и ContractTime на специальную строку 'nan' (листинг 27).

2.5. Применить DictVectorizer для получения one-hot-кодирования признаков LocationNormalized и ContractTime.

2.6. Объединить все полученные признаки в одну матрицу «объекты-признаки».

Обратите внимание!

- *Матрицы для текстов и категориальных признаков являются разреженными. Для объединения их столбцов нужно воспользоваться функцией scipy.sparse.hstack.*

2.7. Обучить гребневую регрессию с параметрами alpha=1 и random_state=241, помня, что целевая переменная записана в столбце SalaryNormalized. Обучение производится вызовом метода fit.

3. Построить прогноз для двух примеров из файла salary-test-mini.csv. Прогноз строится методом predict.

Код решения задачи (с комментариями) – в листинге 28. Результат работы – два прогнозных числа для двух записей из тестовой выборки: [56876.4573163 37557.00551031].

Листинг 28

```
# -*- coding: utf-8 -*-
import pandas
from sklearn.feature_extraction import DictVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Ridge
from scipy.sparse import hstack

#Загрузка данных
data_train = pandas.read_csv('salary-train.csv')
data_test = pandas.read_csv('salary-test-mini.csv')

# Приведение текста к нижнему регистру
data_train['FullDescription'] = data_train.FullDescription.str.lower()
data_test['FullDescription'] = data_test.FullDescription.str.lower()
```

Окончание листинга 28

```
# Удаление ненужных символов
data_train['FullDescription'] = data_train['FullDescription'].replace('[^a-zA-Z0-9]', ' ',
regex=True)
data_test['FullDescription'] = data_test['FullDescription'].replace('[^a-zA-Z0-9]', ' ',
regex=True)
# Преобразование текста в вектор признаков, используя TdifVectorizer из sklearn
vectorizer = TfidfVectorizer(min_df=10)
train_text_feature_matrix = vectorizer.fit_transform(data_train['FullDescription'])
test_text_feature_matrix = vectorizer.transform(data_test['FullDescription'])
# Заполнение пустых ячеек
data_train['LocationNormalized'].fillna('nan', inplace=True)
data_train['ContractTime'].fillna('nan', inplace=True)
data_test['LocationNormalized'].fillna('nan', inplace=True)
data_test['ContractTime'].fillna('nan', inplace=True)
# Преобразование категориальных признаков в числовые
enc = DictVectorizer()
train_dic = data_train[['LocationNormalized', 'ContractTime']].to_dict('records')
test_dic = data_test[['LocationNormalized', 'ContractTime']].to_dict('records')
X_train_categ = enc.fit_transform(train_dic)
X_test_categ = enc.transform(test_dic)
# Здесь по горизонтали объединяется разреженная матрица с признакими текста (из
#столбца FullDescription) и матрица с закодированными категориями (из столбцов
#Location Normalized и Contract Time)
x_train = hstack((train_text_feature_matrix, X_train_categ))
y_train = data_train['SalaryNormalized'].values
x_test = hstack((test_text_feature_matrix, X_test_categ))
#Создание и обучение регрессии
ridge_regression = Ridge(alpha=1, random_state=241)
ridge_regression.fit(x_train, y_train)
#Получение и распечатка прогнозных значений
y_test = ridge_regression.predict(x_test)
print y_test
```

Сохранение и загрузка обученной модели

Как вы могли заметить, работа программы идет довольно долго. Большую часть этого времени занимает процесс обучения, поэтому при работе с обученной моделью используют практику сохранения и загрузки ее параметров в\из файл(а). Рассмотрим следующий

пример. Пусть у нас есть некоторый массив данных о двух факторах и зависимой переменной, сохраненные в файле «data-logistic.csv» (скачать можно отсюда: [41]). Выполним следующее: обучим на этих данных модель линейной регрессии, сохраним ее в файл, создадим еще одну новую модель линейной регрессии, обучим ее на части данных, затем загрузим ранее обученную модель из файла и сравним результаты предсказания обеих моделей. Сохранение и загрузку мы будем выполнять с помощью модуля pickle, и для удобства вынесем код загрузки и сохранения в отдельные функции LoadFromObject и SaveToObject. Решение задачи показано в листинге 29, а ее результат – на рисунке 59.

Листинг 29

```
# -*- coding: utf-8 -*-

import pandas
import pandas.core.series as Series
import numpy as np
from scipy.sparse import hstack
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
import sys
import matplotlib.pyplotasplt
from sklearn import linear_model
import pickle

#Определяем функцию сохранения
def SaveToObject(obj, filename):
  with open(filename, 'wb') as output:
  pickle.dump(obj, output)

#Опредлеляем функцию загрузки
def LoadFromObject(filename):
   f = open(filename, 'rb')
   loaded_obj = pickle.load(f)
   f.close()
   return loaded_obj
```

Окончание листинга 29

```
# Загружаем данные
data_train = pandas.read_csv('data-logistic.csv')
X = data_train.ix[:,1:3].values
y = (data_train.ix[:,0].values + 1)/2
#Для лучешго понимания представления данных распечатайте формы массивов:
print X.shape
print y.shape
# Создание, обучение модели и получение ее прогноза
regression = linear_model.LogisticRegression()
regression.fit(X, y)
ans = regression.predict(X)
# Оценка качества созданной модели
metric = roc_auc_score(y, ans)
print "AUC Trained: ",  metric.real
#Сохранение и загрузка модели в файл с именем model
SaveToObject(regression, "model")
loaded_model = LoadFromObject("model")
#Создание, обучение новой модели, получение ее прогноза и оценка качества
not_fitted = linear_model.LogisticRegression()
not_fitted.fit(X[0:2,:], y[0:2])
ans = not_fitted.predict(X)
metric = roc_auc_score(y, ans)
print "AUC not Trained: ", metric.real
#получение прогноза и оценки качества загруженной модели
ans = loaded_model.predict(X)
metric = roc_auc_score(y, ans)
print "AUCLoaded: ", metric.real
```

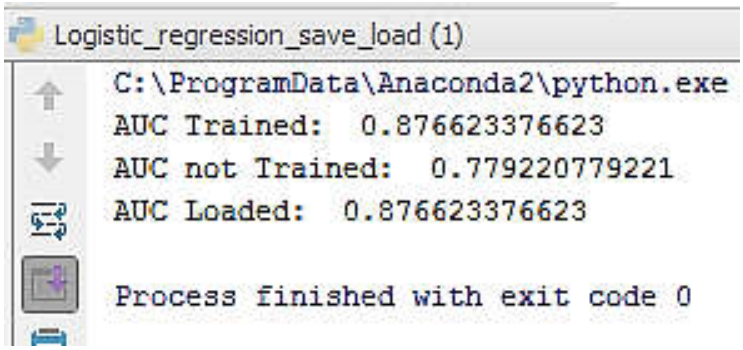


Рисунок 59. Результат работы кода из листинга 29

Как видно из рисунка 59, модель, обученная на части данных, дала более плохой результат, тогда как качество загруженной модели не изменилось. То есть можно говорить о том, что сохранение и загрузка прошли корректно.

Работа с логистической регрессией

Мы рассмотрели применение регрессии для решения задачи предсказания, теперь перейдем к задаче классификации и познакомимся с еще одной моделью – логистической регрессией. Это один из видов линейных классификаторов. Ее особенность заключается в том, что результатом является вероятность принадлежности объекта к классу N, тогда как большинство линейных классификаторов могут выдавать только номера классов.

Логистическая регрессия использует достаточно сложную функцию для оценки качества, которая не допускает записи решения в явном виде (в отличие от, например, линейной регрессии). Тем не менее логистическую регрессию можно настраивать с помощью градиентного спуска.

Для примера мы выполним следующее: сгенерируем некоторую случайную выборку и каждому ее элементу поставим в соответствие число 0 или 1, которое будет обозначать принадлежность к первому или второму классу. Затем обучим логистическую модель на этих данных и линейную модель. После – построим график и сравним результаты. Для генерации случайной выборки из 100 элементов используем следующий код из листинга 30.

Листинг 30

```
import numpy as np
…..
n_samples = 100
np.random.seed(0)
X = np.random.normal(size=n_samples)
```

Обратите внимание!

- *В дальнейших листингах операторы импорта библиотек будут появляться только при первом вызове импортируемых элементов, так как предполагается, что все представленные ниже листинги формируют единый код, который вам предлагается собрать в единый файл самостоятельно.*

Для формирования множества откликов Y определим функцию, реализующую правило: Y=1 для всех X>0 и Y=0 – в остальных случаях. Код функции можно увидеть в листинге 31.

Листинг 31

```
Y = (X > 0).astype(np.float)
```

Следующим шагом нам необходимо выполнить преобразования нашего одномерного массива X: во-первых, добавить небольшой разброс и смещение, чтобы избавиться от прямой функциональной зависимости. Во-вторых, нужно преобразовать его в двумерный массив. Последнее выполняется для функции обучения регрессии: на вход она требует данные в формате двумерного массива. Преобразование выполняется кодом, представленным листинге 32.

Листинг 32

```
X[X > 0] *= 4
X += .3 * np.random.normal(size=n_samples)
X = X[:, np.newaxis]
```

Теперь мы можем объявить и обучить модели регрессии, как показано в листинге 33.

Листинг 33

```
from sklearn import linear_model
log_reg = linear_model.LogisticRegression()
log_reg.fit(X, Y)
lin_reg = linear_model.LinearRegression()
lin_reg.fit(X, Y)
```

Подробнее о реализации логистической модели можно прочитать здесь: [44], а линейной – здесь: [45].

Следующим шагом нам необходимо сгенерировать тестовые данные и выполнить предсказание. Последнее будет выполняться методом predict. Он возвращает бинарные метки классов (0,1). Кроме этого существует еще метод predict_proba (для логистической регрессии). Он возвращает вероятность принадлежности объекта к классу 0 или 1 соответственно. Поскольку у нас два класса, то в результате predict_proba получается двумерный массив (один столбец отражает принадлежность к классу 0, а другой к классу 1). Для отображения мы выберем лишь первый столбец.

Обратите внимание!

- *Сумма элементов массивов (столбцов, векторов) с одним индексом равна единице по правилу полной вероятности.*

Код генерации тестовых данных и вызовы методов прогноза представлен в листинге 34.

Листинг 34

```
X_test = np.linspace(start=-5, stop=10, num=300)
X_test = X_test[:, np.newaxis]
y_log_label = log_reg.predict(X_test)
y_lin = lin_reg.predict(X_test)
y_log_probabilty = log_reg.predict_proba(X_test)[:,1]
```

Следующим шагом нам необходимо вывести график для отображения результатов. Создавать полотно для рисования графиков и отображать их мы будем с помощью библиотеки Matplotlib, как показано в листинге 35.

Листинг 35

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
....
```

Окончание листинга 35

```
# рисуем исходные точки
plt.figure(1, figsize=(4, 3))
plt.scatter(X.ravel(), y, color='black', zorder=20)
# рисуем график меток логистической регрессии
plt.plot(X_test, y_log_label, color='red', linewidth=3)
# рисуем график линейной регрессии
plt.plot(X_test, y_lin, linewidth=1)
# добавляем линию уровня, по которому точки будут относится к 1 или 2 классу
plt.axhline(.5, color='green')
# рисуем логистическую кривую для вероятностей принадежности объектов к
#классам
plt.plot(X_test, y_log_probabilty, color='blue', linewidth=3)
# настраиваем оси и выводим график
plt.ylabel('y')
plt.xlabel('X')
plt.xticks(range(-5, 10))
plt.yticks([0, 0.5, 1])
plt.ylim(-.25, 1.25)
plt.xlim(-4, 10)
plt.legend(('LabelLogisticRegressionModel', 'LinearRegressionModel',
'ProbabiltyLogRegression'), loc="lowerright", fontsize='small')
plt.show()
```

В результате после выполнения кода появится график, как показано на рисунке 60.

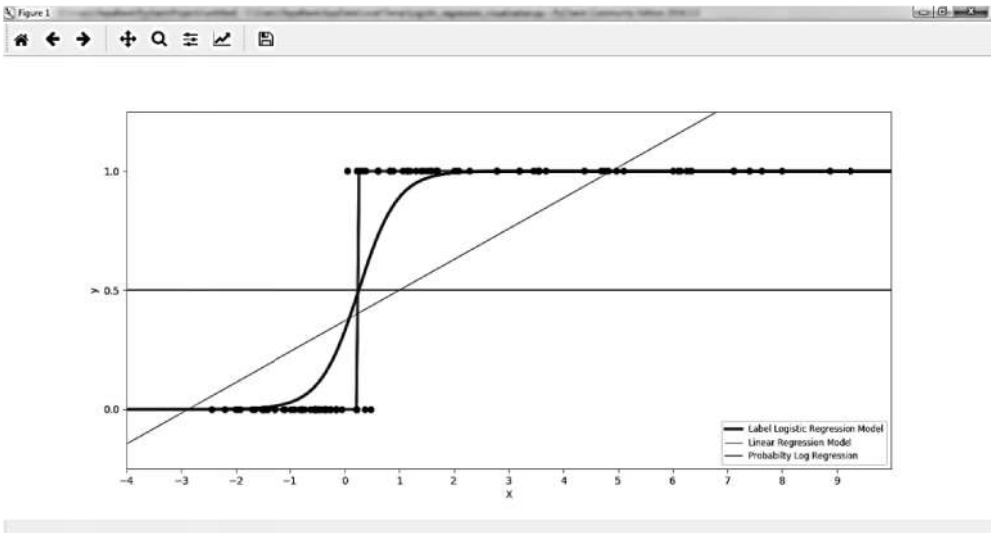


Рисунок 60. Графики регрессий

Для лучшего понимания вам рекомендуется воспроизвести код и посмотреть на график, который вам выведет программа. На нем должно быть видно, что красная кривая (результат логистического предсказания\классификации) лучше приближает распределение точек (лишь несколько точек вылетает). Красная линия построена лишь по меткам, полученным от log_reg.predict, поэтому она имеет такой резкий переход. Такой «округленный» результат легко использовать в дальнейшем, хотя в таком случае мы не получаем информацию о вероятностях. Линейная регрессия (голубая линия) приближает данные не так хорошо.

Важно понимать, что линейная регрессия отнесет все точки (x<1) к классу 0, так как результат функции линейной регрессии будет меньше 0.5. А все точки (X>1) будут отнесены к классу 1, так как результат функции линейной регрессии будет больше 0.5. Для этого на графике выведена разделяющая линия Y=0.5. Граница разделения классов по логистической регрессии проходит примерно в точке X=0.25, что разделяет классы гораздо точнее. Толстая синяя линия отображает вероятностную логистическую кривую, которая дает больше детальной информации о предсказании. Поскольку результат представляет собой не бинарные метки классов, а вероятность принадлежности к выбранном классу, то кривая получается гладкой.

Для точной оценки качества моделей воспользуемся функциями score и accuracy_score в scikit-learn. Поскольку линейная регрессия выдает вещественные значения, а y_test сгенерирована как бинарный массив, то для оценки качества нужно использовать функцию model.score(), которая способна читать и бинарные, и вещественные ответы. Для подсчета качества логистической регрессии можно воспользоваться измерителем качества классификации accuracy_score. Но перед этим нам необходимо получить реальную принадлежность данных тестовой выборки к классам. Сделать это можно с помощью определенной нами функции для обучающей выборки, как показано в листинге 36.

Листинг 36

```
y_test = (X_test > 0).astype(np.float)
```

Теперь можно вывести значения качества классификации для обеих моделей, как показано в листинге 37. Для линейной регрессии это выполняется ее методом score, а для логистической – методом accuracy_score из sklearn.metrics.

Листинг 37

```
from sklearn.metrics import accuracy_score
....
# оцениваем точность решений
lin_score = lin_reg.score(X_test, y_test)
print "Linear regression accuracy: ", lin_score
log_score = accuracy_score(y_log_label, y_test)
print "Logistic regression accuracy: ", log_score
```

В итоге в консоль будут выведены следующие результаты: Linearregression accuracy=0,52442, Logisticregression accuracy=0,98333. Они подтверждают, что логистическая регрессия выполнила классификацию более качественно.

Решение задачи ранжирования признаков

Следующее, что нам необходимо рассмотреть, – применение регрессионных моделей для определения важности признаков. Здесь в качестве примера возьмем регрессионную проблему Фридмана, когда на вход моделей подается 14 факторов, выход рассчитывается по формуле, использующей только пять факторов, но факторы 1-5, а также 10-14 взаимозависимы. Наша задача – посмотреть, как разные виды регрессий оценят важности факторов и какой из них будет иметь наибольшую среднюю значимость по всем моделям. Для работы мы возьмем три модели: линейную регрессию, гребневую и лассо (линейную регрессию с L1-регуляризатором). О реализации в scikit-learn последней модели можно подробнее прочитать здесь: [46].

Первое, что нам необходимо сделать для решения поставленной задачи, – подключить необходимые модули, как в листинге 38. Вторая строка кода из листинга 38 – импорт класса, выполняющего масштабирование данных до заданного диапазона (например, так, чтобы все значение находились в диапазоне от 0 до 1). Подробнее о реализации класса можно прочитать здесь: [47]. Необходимость его использования объясняется следующим: каждая модель регрессии дает оценки важности признаков в своем диапазоне. Для того чтобы найти признак с максимальной средней важностью по трем моделям, нам необходимо привести выданные ими оценки к одному виду, в чем и поможет нам указанный класс.

Листинг 38

```
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import MinMaxScaler
import numpy as np
```

После импорта необходимых классов мы должны сгенерировать исходные данные, как показано в листинге 39.

Листинг 39

```
#генерируем исходные данные: 750 строк-наблюдений и 14 столбцов-признаков
np.random.seed(0)
size = 750
X = np.random.uniform(0, 1, (size, 14))
#Задаем функцию-выход: регрессионную проблему Фридмана
Y = (10 * np.sin(np.pi*X[:,0]*X[:,1]) + 20*(X[:,2] - .5)**2 +
    10*X[:,3] + 5*X[:,4]**5 + np.random.normal(0,1))
#Добавляем зависимость признаков
X[:,10:] = X[:,:4] + np.random.normal(0, .025, (size,4))
```

Теперь создаем и обучаем модели (листинг 40).

Листинг 40

```
#линейная модель
lr = LinearRegression()
```

Окончание листинга 40

```
lr.fit(X, Y)
#гребневая модель
ridge = Ridge(alpha=7)
ridge.fit(X, Y)
#Лассо
lasso = Lasso(alpha=.05)
lasso.fit(X, Y)
```

Наконец, нам нужно выгрузить в единый массив размера 3×14 (количество_моделей и количество_признаков) все оценки моделей по признакам. Найти средние оценки и вывести результат в формате списка пар {номер_признака – средняя_оценка}, отсортированном по убыванию. Получить оценки признаков можно через поле coef_ у каждой из наших моделей. Для удобства отображения данных поместим их в конструкцию вида: [имя_модели : [{имя_признака : оценка},{имя_признака : оценка}]]. То есть верхним уровнем у нас будет словарь, где ключом будет имя модели. В нем будут располагаться три записи из четырнадцати пар каждая. Пример содержимого списка представлен в листинге 41.

Листинг 41

```
{'Lasso': {'x13': 0.0, 'x14': 0.0, 'x10': 0.0, 'x8': 0.0, 'x9': 0.0, 'x11': 0.0, 'x2': 0.72, 'x3': 0.0,
'x12': 0.0, 'x1': 0.69, 'x6': 0.0, 'x7': 0.0, 'x4': 1.0, 'x5': 0.29},
'Ridge': {'x13': 0.0, 'x14': 0.92, 'x10': 0.01, 'x8': 0.08, 'x9': 0.0, 'x11': 0.59, 'x2': 0.75, 'x3': 0.06,
'x12': 0.67, 'x1': 0.76, 'x6': 0.08, 'x7': 0.03, 'x4': 1.0, 'x5': 0.61},
'Linear reg': {'x13': 0.48, 'x14': 0.12, 'x10': 0.01, 'x8': 0.03, 'x9': 0.0, 'x11': 0.59, 'x2': 0.61, 'x3':
0.51, 'x12': 0.19, 'x1': 1.0, 'x6': 0.02, 'x7': 0.0, 'x4': 0.69, 'x5': 0.19}}
```

Первое, что нам необходимо сделать, чтобы реализовать описанное, – создать список вида ['x1', 'x2', 'x3', 'x4', 'x5', 'x6', 'x7', 'x8', 'x9', 'x10', 'x11', 'x12', 'x13', 'x14'], содержащий имена признаков. Выполнить это можно с помощью кода из листинга 42.

Листинг 42

```
names = ["x%s" % i for i in range(1,15)]
```

Затем нам необходимо объявить функцию, которая на вход будет получать сформированный выше список и список оценок по признакам. Выходом же функции должен быть словарь, составленный из попарно соотнесенных элементов списков (то есть словарь из пар имя_признака: оценка_признака, где первое – ключ, а второе – значение). Причем оценки внутри функции должны быть приведены к единому диапазону от 0 до 1 и округлены до сотых.

В теле функции нам потребуются следующие объекты. Во-первых, класс MinMaxScaler, о котором было сказано выше. Во-вторых, функция abs пакета Numpy для получения абсолютных значений оценок(модуля). В-третьих, функция map, позволяющая применить операцию округления (round) к каждому элементу массива посредствам оператора lambda. В-четвертых, нам понадобятся функции rashape и ravel. Первая – для переформирования входного массива признаков из одной строки и четырнадцати столбцов в четырнадцать строк и один столбец, вторая – для преобразования полученного двумерного массива в одномерный. В-пятых, нам потребуются функции zip и dict. Первая попарно соотнесет имена признаков и оценки, вторая – преобразует полученные пары в словарь. Примеры использования функций zip, map и lambda можно найти здесь: [48]. Код реализации описанных действий можно найти в листинге 43.

Листинг 43

```
def rank_to_dict(ranks, names):
    ranks = np.abs(ranks)

    minmax = MinMaxScaler()

    ranks = minmax.fit_transform(np.array(ranks).reshape(14,1)).ravel()
    ranks = map(lambda x: round(x, 2), ranks)
    return dict(zip(names, ranks))
```

Вызов функции для наших моделей представлен в листинге 44.

Листинг 44

```
ranks["Linear reg"] = rank_to_dict(lr.coef_, names)
ranks["Ridge"] = rank_to_dict(ridge.coef_, names)
ranks["Lasso"] = rank_to_dict(lasso.coef_, names)
```

Последним шагом мы должны сформировать среднее по каждому признаку, загрузив их в отдельный список. Затем отсортировать его по убыванию и вывести. Реализовать это можно, используя код из листинга 45.

Листинг 45

```
#Создаем пустой список для данных
mean = {}

#«Бежим» по списку ranks
for key, value in ranks.iteritems():
#«Пробегаемся» по списку значений ranks, которые являются парой имя:оценка
for item in value.iteritems():
#имя будет ключом для нашего mean
#если элемента с текущим ключем в mean нет - добавляем
if(item[0] not in mean):
mean[item[0]] = 0

#суммируем значения по каждому ключу-имени признака
      mean[item[0]] += item[1]

#находим среднее по каждому признаку
for key, value in mean.iteritems():
   res=value/len(ranks)
   mean[key] = round(res, 2)

#сортируем и распечатываем список
mean = sorted(mean.iteritems(), key=lambda (x, y): y, reverse=True)

print "MEAN"
print mean
```

Результат работы программы представлен в листинге 46.

Листинг 46

```
MEAN
[('x4', 0.9), ('x1', 0.82), ('x2', 0.69), ('x11', 0.39), ('x5', 0.36), ('x14', 0.35), ('x12', 0.29), ('x3', 0.19), ('x13', 0.16), ('x8', 0.04), ('x6', 0.03), ('x10', 0.01), ('x7', 0.01), ('x9', 0.0)]
```

Мы видим, что по средней оценке наиболее важным оказался четвертый признак, затем первый и потом второй. Сравним эти данные с показателями каждой отдельной модели. Для этого отсортируем массив ranks также по убыванию оценок и выведем его, используя код из листинга 47. Результат работы кода представлен в листинге 48.

Листинг 47

```
for key, value in ranks.iteritems():
    ranks[key] = sorted(value.iteritems(), key=lambda (x, y): y, reverse=True)
for key, value in ranks.iteritems():
    print key
    print value
```

Листинг 48

```
Lasso
[('x4', 1.0), ('x2', 0.72), ('x1', 0.69), ('x5', 0.29), ('x13', 0.0), ('x14', 0.0), ('x10', 0.0), ('x8', 0.0), ('x9', 0.0), ('x11', 0.0), ('x3', 0.0), ('x12', 0.0), ('x6', 0.0), ('x7', 0.0)]
Ridge
[('x4', 1.0), ('x14', 0.92), ('x1', 0.76), ('x2', 0.75), ('x12', 0.67), ('x5', 0.61), ('x11', 0.59), ('x8', 0.08), ('x6', 0.08), ('x3', 0.06), ('x7', 0.03), ('x10', 0.01), ('x13', 0.0), ('x9', 0.0)]
Linear reg
[('x1', 1.0), ('x4', 0.69), ('x2', 0.61), ('x11', 0.59), ('x3', 0.51), ('x13', 0.48), ('x12', 0.19), ('x5', 0.19), ('x14', 0.12), ('x8', 0.03), ('x6', 0.02), ('x10', 0.01), ('x9', 0.0), ('x7', 0.0)]
```

Как можно увидеть, метод Lasso отобрал признаки x1, x2, x4, x5 как значимые параметры, а все остальные совсем незначимые (так как они равны нулю). Что довольно близко к истине, так как параметры x1-x5 заданы как независимые случайные величины. Но взвешивание по такому методу довольно грубое, резкое.

Метод Ridge сделал более мягкое взвешивание. Но несмотря на то, что x3 также упущен, его влияние может быть учтено через скорре-

лированные параметры x14, x12, x11 (которые были также отобраны как важные). Поэтому данный метод может быть очень полезным.

Метод линейной регрессии также отдал предпочтение многим действительно важным параметрам (причем x3 здесь имеет гораздо больший вес).

В заключение можно сказать, что учет оценок от разных методов позволит более оптимально выделить значимые признаки и отделить от менее значимых. Но, конечно, без реальных экспериментов, проверяющих выбранное множество параметров, невозможно гарантировать идеального разбиения для всех задач.

Кроме рассмотренных выше моделей для отбора признаков еще можно использовать:

1. RandomizedLasso из sklearn.linear_model (оценки признаков будут находиться в поле scores_). Это так называемое рандомизированное лассо работает, разделяя тренировочные данные на подвыборки и вычисляя Лассо-оценки признаков, где штраф случайного подмножества коэффициентов масштабирован. Применяя такую двойную рандомизацию несколько раз, метод присваивает наивысшие оценки признакам, выбор которых повторялся в процессе рандомизации. Это называется «выбором стабильности». Если же сказать короче, то признаки, которые выбираются наиболее часто, считаются значимыми. Подробнее об этом классе можно почитать здесь: [49].

2. RFE из sklearn.feature_selection (оценки признаков будут находиться в поле ranking_, однако в отличие от всех остальных моделей, класс выдает не веса при коэффициентах регрессии, а именно ранг для каждого признака. Так наиболее важные признаки будут иметь ранг – «1», а менее важные признаки ранг больше «1». Коэффициенты остальных моделей тем важнее, чем больше их абсолютное значение). Класс RFE выполняет ранжирование признаков через рекурсивный отбор путем отсева признаков. Целью рекурсивного устранения элемента (RFE) является выбор признаков путем рекурсивного рассмотрения меньших и меньших наборов признаков, учитывая внешнюю оценку, которая присваивает признакам веса

(например, коэффициенты линейной модели). Сначала оценщик обучается на начальном наборе признаков и ассоциированных с ними весами. Затем признаки, абсолютные веса которых являются наименьшими, удаляются из текущего набора. Эта процедура рекурсивно повторяется на оставшемся множестве до тех пор, пока в конечном итоге не будет достигнуто желаемое количество признаков. То есть работа с этим классом строится, как показано в листинге 49. Подробнее о нем можно почитать здесь: [50].

Листинг 49

```
lr = LinearRegression()
lr.fit(X, Y)

rfe = RFE(lr)
rfe.fit(X,Y)
```

3. RandomForestRegressor из sklearn.ensemble (оценки признаков будут находиться в поле feature_importances_). Регрессор на основе Случайного леса – это мета-оценщик, который обучает несколько классификационных деревьев решений на разных подвыборках данных и использует усреднение для улучшения точности и контроля переобучения. Размер подвыборки всегда соответствует размеру входной выборки, но образцы из выборки перемешиваются (если параметр bootstrap=True, что является значением по умолчанию). Подробнее об этом классе можно посмотреть здесь: [51].

4. f_regression из sklearn.feature_selection (для получения оценок см.код из листинга 50. Оценки будут находиться в объекте f). Это быстрая линейная модель, позволяющая оценить эффект одного признака и применяющаяся последовательно к множеству признаков. В основе ее работы лежит вычисление взаимной корреляции между каждым признаком и выходной переменной. Более подробно можно посмотреть здесь: [52].

Листинг 50

```
f, pval  = f_regression(X, Y, center=True)
```

Работа с полиномиальной регрессией

Напоследок, заканчивая разговор о регрессионных моделях, мы познакомимся с полиномиальной регрессией, а также посмотрим на практике такое явление, как переобучение модели. Рассматривать мы это будем на следующем примере: сгенерируем точки функцией косинуса и попробуем аппроксимировать их линейной регрессией, полиномиальной регрессией со степенью 4 и 15, а также гребневой регрессией и созданными на основе нее полиномами 4 и 15 степеней. Затем рассчитаем точность моделей, выведем шесть графиков и сравним результаты.

Итак, первым шагом создадим обычную функция косинуса от аргумента, записанную в виде lambda-выражения (листинг 51). Особенности Python позволяют использовать в качестве аргумента не только числа, но и массивы\векторы (так что использование циклов не требуется).

Листинг 51

```
import numpy as np
true_fun = lambda X: np.cos(1.5 * np.pi * X)
```

Представленная в листинге 51 функция будет использоваться как истинная закономерность в некоторых данных. То есть именно то, что аналитики и ученые пытаются выяснить, проводя измерения некоторого объекта природы или бизнеса. Собственно, поэтому название функции – true_fun. В дальнейшем эту функцию можно будет вызывать подобно функции, которая определена через def, как показано в листинге 52.

Листинг 52

```
result= true_fun(something_data)
```

Далее, для корректного проведения эксперимента необходимо создать некую выборку, которая будет служить аналогом измерений истинной закономерности. Для этого мы создадим распределение\выборку

на основе объявленной ранее функции с небольшим случайным смещением по X и Y (листинг 53).

Листинг 53

```
np.random.seed(0)
n_samples = 30
X = np.sort(np.random.rand(n_samples))
y = true_fun(X) + np.random.randn(n_samples) * 0.1
```

Полиномиальная регрессия задается в несколько шагов. Сперва необходимо определить коэффициенты полинома (которые автоматически генерируются с помощью функции PolynomialFeatures) и модель для основы (в нашем случае линейную или гребневую). Затем необходимо соединить вместе базовую регрессию и полиномиальные коэффициенты (вектор или матрицу преобразования) как последовательность трансформаций некоторых данных. Итоговый объект и будет являться полиномиальной моделью. Трансформацию данных можно выполнить с помощью объекта Pipeline из sklearn.pipeline (подробнее о нем можно почитать здесь: [53]).

Для удобства работы и сокращения кода сделаем следующее: объявим массив с нужными нам степенями полинома, как показано в листинге 54.

Листинг 54

```
degrees = [1, 4, 15]
```

Затем объявим функцию, которая будет принимать в качестве параметров номер степени (индекс элемента в массиве dergees), полотно, на котором нужно отобразить график, базовую модель и уровень графика на полотне (первый или второй ряд по вертикали). Внутри тела функции будет создаваться и обучаться нужная полиномиальная модель, результат работы которой будет выведен на возвращаемом функцией графике. Кроме точек на графике будет отображена точность модели, вычисленная методом кросс-валидации:

средние оценки качества по метрике «neg_mean_squared_error» (среднеквадратичная ошибка). Эти оценки будут получены с помощью объекта cross_val_score из sklearn.model_selection. Подробнее о кросс-валидации в scikit можно почитать здесь: [54, 55].

Объявление функции показано в листинге 56, с работой которого вам предлагается разобраться самостоятельно. Единственно, поясним назначение параметра cv в cross_val_score. Параметр cv определяет, на сколько групп будет разбита тренировочная выборка. Каждая группа будет содержать тренировочную часть А и тестовую Б. Причем эти множества (А и Б) не будут пересекаться. Так для следующих данных: входа X = np.array([[11, 2], [3, 14], [1, 2], [3, 4]]), выхода y = np.array([1, 2, 3, 4]) cross_val_score с параметром cv=2 даст следующее разбиение на 2 группы\итерации (листинг 55).

Листинг 55

```
TRAIN: [2 3] TEST: [0 1]
TRAIN: [0 1] TEST: [2 3]
```

Массив TRAIN содержит это индексы тех объектов, которые входят в тренировочную выборку. То есть TRAIN: [2 3] означает, что в тренировочную выборку войдут второй и третий объекты из X: [1,2], [3, 4]. Все остальные – в тестовую.

Листинг 56

```
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.model_selection import cross_val_score
def MakeExample(index, plt, model,k):
   polynomial_features = PolynomialFeatures(degree=degrees[index],
                                            include_bias=False)
```

Окончание листинга 56

```
    pipeline = Pipeline([("polynomial_features", polynomial_features),
                         ("linear_regression", model)])
    pipeline.fit(X[:, np.newaxis], y)
    scores = cross_val_score(pipeline, X[:, np.newaxis], y,
                             scoring="neg_mean_squared_error", cv=10)
    X_test = np.linspace(0, 1, 100)
    plt.plot(X_test, pipeline.predict(X_test[:, np.newaxis]), label="Model")
    plt.plot(X_test, true_fun(X_test), label="True function")
    plt.scatter(X, y, label="Samples")
    if k == 1:
      plt.xlabel("x")
    plt.ylabel("y")
    plt.xlim((0, 1))
    plt.ylim((-2, 2))
    plt.legend(loc="best")
    if k==0:
     plt.title("Degree {}\nMSE = {:.2e}(+/- {:.2e})".format(degrees[i], -scores.mean(),
scores.std()))
    else:
     plt.title("MSE = {:.2e}(+/- {:.2e})".format(-scores.mean(), scores.std()))
    return plt
```

Код вызова функции из листинга 56 (и, соответственно отрисовка графиков) представлен в листинге 57, а результат работы кода – на рисунке 61.

Листинг 57

```
plt.figure(figsize=(14, 6))
for i in range(len(degrees)):
    linear_regression = LinearRegression()
    plt.subplot(2, len(degrees), i+1)
    plt = MakeExample(i, plt, linear_regression,0)
for i in range(len(degrees)):
    ridge = Ridge(alpha=0.02)
    plt.subplot(2, len(degrees), len(degrees) + i + 1)
    plt = MakeExample(i, plt, ridge,1)
plt.show()
```

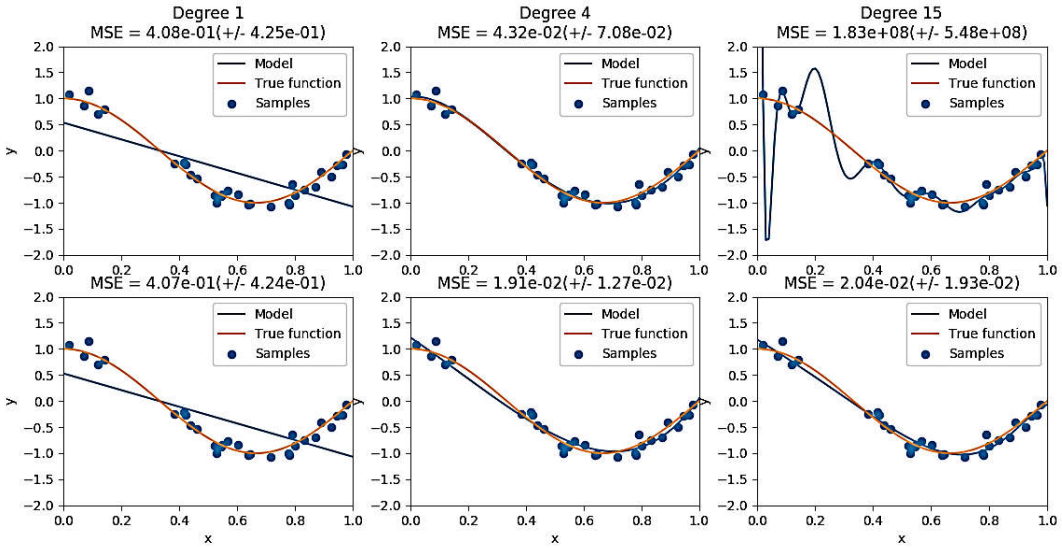


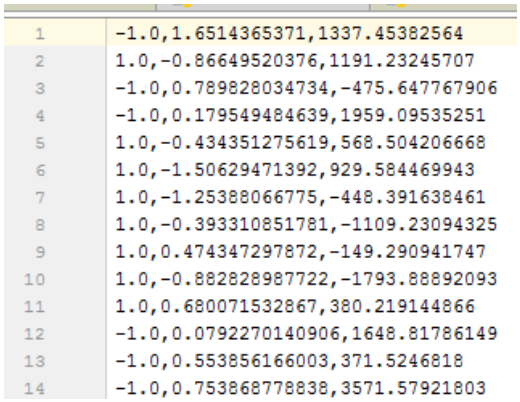
Рисунок 61. Результаты работы с разными моделями

Для лучшего понимания вам рекомендуется воспроизвести код и посмотреть на рисунок, выданный вашей программой. Если говорить о моделях, построенных на основе линейной регрессии, то, сравнивая первый и второй графики (полином 1 и 4 степени), можно сделать вывод, что повышение степени положительно влияет на качество модели. Но третий график (полином 15 степени) демонстрирует большую ошибку. Как мы видим, на тренировочной выборке модель показала хороший результат (кривая проходит через все точки выборки). Но на тестовой выборке модель «ведет себя плохо» (плохо приближает true_fun). Это и означает, что при повышении сложности модели есть риск перейти за критическую точку и переобучить модель. Однако, как мы видим на графиках гребневой регрессии, регуляризация позволяет улучшить качество решения.

Работа с простейшими моделями нейронных сетей

Следующее, с чем нам необходимо познакомиться, – работа с простейшими моделям нейронных сетей, которые содержатся в библиотеке Sklearn (в дальнейшем будет рассмотрена работа с другими библиотеками, которые содержат более сложные алго-ритмы и позволяют создавать другие нейроклассификаторы и модели). Однако Sklearn не специализируется на нейросетевых моделях и не содержит более сложные алгоритмы.

Начнем мы с обычного персептрона, являющегося одним из вариантов линейных моделей. Это распространенный класс моделей, которые отличаются своей простотой и скоростью работы. Их можно обучать за разумное время на очень больших объемах данных, и при этом они могут работать с любыми типами признаков: вещественными, категориальными, разреженными. Рассмотрим решение следующей задачи: пусть у нас есть данные вида, показанного на рисунке 62, и сохраненные в файлах perceptron-train.csv и perceptron-test.csv (скачать файлы можно отсюда: [41]). Целевая переменная записана в первом столбце, признаки – во втором и третьем. Выполним следующее: применим к этим данным однослойный персептрон (о реализации модели Perceptron более подробно здесь: [56]), многослойный персептрон (о реализации модели MLPClassifier более подробно здесь: [57]), сравним точности предсказания моделей, затем нормализуем данные и повторим сравнение. Причем данные мы будем прогонять 100 раз, и на каждом прогоне выполнять 2000 эпох обучения. В качестве показателей выведем минимальные, максимальные, средние значения accuracy и стандартное отклонение, полученные на 100 прогонах. А также выведем график, визуализирующий результаты.



```
1   -1.0,1.6514365371,1337.45382564
2   1.0,-0.86649520376,1191.23245707
3   -1.0,0.789828034734,-475.647767906
4   -1.0,0.179549484639,1959.09535251
5   1.0,-0.434351275619,568.504206668
6   1.0,-1.50629471392,929.584469943
7   1.0,-1.25388066775,-448.391638461
8   1.0,-0.393310851781,-1109.23094325
9   1.0,0.474347297872,-149.290941747
10  1.0,-0.882828987722,-1793.88892093
11  1.0,0.680071532867,380.219144866
12  -1.0,0.0792270140906,1648.81786149
13  -1.0,0.553856166003,371.5246818
14  -1.0,0.753868778838,3571.57921803
```

Рисунок 62. Данные для тестовой задачи

Поясним необходимость нормализации. Как и в случае с метрическими методами, качество линейных алгоритмов зависит от некоторых свойств данных. В частности, признаки должны иметь одина-

ковый масштаб. Если это не так, и масштаб одного признака сильно превосходит масштаб других, то качество может резко упасть.

Один из способов достижения нужного свойства признаков заключается в их стандартизации. Для этого берется набор значений признака на всех объектах, вычисляется их среднее значение и стандартное отклонение. После этого из всех значений признака вычитается среднее, и затем полученная разность делится на стандартное отклонение. В ходе нашего эксперимента мы выясним, улучшает ли нормализация качество работы модели. Код решения задачи (с комментариями) представлен в листинге 58. Результат программы, выдаваемый в консоль, представлен на рисунке 63, а построенный график – на рисунке 64.

Листинг 58

```
# -*- coding: utf-8 -*-
#Необходимые импорты
import numpy as np
import pandas
from sklearn.linear_model import Perceptron
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.neural_network import MLPClassifier
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
#функция расчета медианы
def median(lst):
    return np.median(np.array(lst))
#Загрузка тренировочных данных
data_train = pandas.read_csv('perceptron-train.csv')
X_train = data_train.ix[:, 1:3].values
y_train = data_train.ix[:, 0].values
#Загрузка тестовых данных
data_test = pandas.read_csv('perceptron-test.csv')
X_test = data_test.ix[:, 1:3].values
y_test = data_test.ix[:, 0].values
```

Продолжение листинга 58

```
#Инициализация массива для счетчика итераций
rs = np.linspace(0,100,num=100)
#Инициализация списков для сохранения accuracy моделей
acc_p = []
acc_pn = []
acc_mlp = []
acc_mlpn = []
#Цикл прогона моделей
for i in rs:
    i = int(i)
#Распечатка номера итерации
   print "Random: ", i
#Создание модели персептрона
    clf = Perceptron(random_state=i, alpha=0.01, n_iter=2000)
#Обучение модели
   clf.fit(X_train, y_train)
#Получение прогноза
    predictions = clf.predict(X_test)
# Расчет показателя accuracy
    acc = accuracy_score(y_test, predictions)
#Распечатка результата
    print "Perceptron: ", acc
#Добавление оценки в список оценок для модели персептрона
    acc_p.append(acc)
#Нормализация данных
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)
#Работа пресептрона с нормализованными данными
    clf = Perceptron(random_state=i, alpha=0.01, n_iter=2000)
    clf.fit(X_train_scaled, y_train)
    predictions = clf.predict(X_test_scaled)
    acc = accuracy_score(y_test, predictions)
    print "Perceptron with normalization: ", acc
    acc_pn.append(acc)
#Создание многослойного классификатора
    mlp = MLPClassifier(random_state=i, solver="sgd", activation="tanh", alpha=0.01,
hidden_layer_sizes=(2, ), max_iter=2000, tol=0.00000001)
```

Окончание листинга 58

```
    mlp.fit(X_train, y_train)
#Работа с ненормализованными данными
    predictions = mlp.predict(X_test)
    acc = accuracy_score(y_test, predictions)
    print "MLP: ",  acc
    acc_mlp.append(acc)
#Работа с нормализованными данными
    mlp = MLPClassifier(random_state=i, solver="sgd", activation="tanh", alpha=0.01,
hidden_layer_sizes=(2, ), max_iter=2000, tol=0.00000001)
    mlp.fit(X_train_scaled, y_train)
    predictions = mlp.predict(X_test_scaled)
    acc = accuracy_score(y_test, predictions)
    print "MLPwith Norm: ",  acc
    acc_mlpn.append(acc)
#Распечатка итоговых результатов
print "Perceptron: ", min(acc_p), median(acc_p), max(acc_p), np.std(acc_p)
print "Perceptron with Norm: ", min(acc_pn), median(acc_pn), max(acc_pn),
np.std(acc_pn)
print "MLP: ", min(acc_mlp), median(acc_mlp), max(acc_mlp), np.std(acc_mlp)
print "MLP with Norm: ", min(acc_mlpn), median(acc_mlpn), max(acc_mlpn),
np.std(acc_mlpn)
#Расчет минимума и максимума для графика
X = np.concatenate((X_train, X_test), axis=0)
x_min, x_max = X[:, 0].min() - .5, X[:, 0].max() + .5
y_min, y_max = X[:, 1].min() - .5, X[:, 1].max() + .5
#Построение графика
figure = plt.figure(figsize=(17, 9))
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
ax = plt.subplot(1, 1, 1)
# Точки из обучающей выборки
ax.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
# Тестовые точки
ax.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright, alpha=0.6)
ax.set_xlim(x_min, x_max)
ax.set_ylim(y_min, y_max)
ax.set_xticks(())
ax.set_yticks(())
plt.show()
```

```
Random:  100
Perceptron:  0.743718592965
Perceptron with normalization:  0.894472361809
MLP:  0.653266331658
MLP with norm:  0.889447236181

Perceptron:  0.35175879397 0.683417085427 0.894472361809 0.130098348038
Perceptron with Norm:  0.251256281407 0.793969849246 0.899497487437 0.12811275683
MLP:  0.346733668342 0.653266331658 0.668341708543 0.0370981920263
MLP with Norm:  0.874371859296 0.889447236181 0.899497487437 0.00389763204616
```

Рисунок 63. Результат работы программы

Воспроизведите код, посмотрите на полученный график. Он должен быть похож на изображенный на рисунке 64. Обратим внимание на один важный технический момент. На графике можно увидеть, что принадлежность к классу (то есть по сути Y) отображается не отдельным измерением, а цветом. Такой механизм позволяет рисовать двумерные графики вместо трехмерных, что проще и нагляднее.

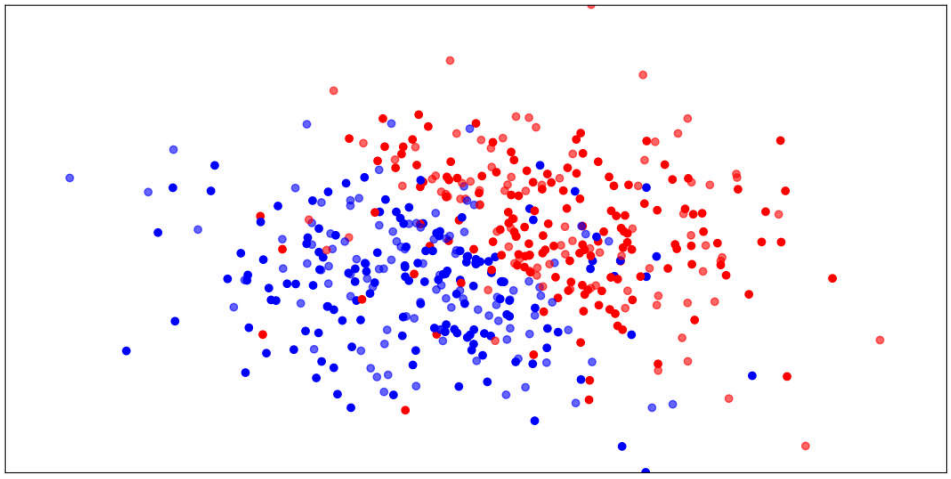


Рисунок 64. Итоговый график

Содержательный анализ результатов работы программы и доработку кода вам предлагается выполнить самостоятельно (см. практическое задание «Многослойный персептрон»).

Реализация алгоритма обучения нейронной сети

Данный пункт посвящен алгоритму обратного распространения ошибки как классическому подходу к обучению нейронных сетей. Поскольку нейронные сети сегодня – это важный класс моделей для построения сложных иерархических признаков, то возникает необходимость детального понимания протекающих в них процессов. Важность именно понимания нейронных сетей обуславливается еще тем, что сложнейшие на сегодняшний день сети еще плохо описаны математически, и требуется проделать много исследовательской работы в этом направлении.

Задача состоит в том, чтобы реализовать обучение полносвязной трехслойной сети прямого распространения на каких-то простых синтетических данных.

Если вы собираетесь заниматься именно исследованием в ML, то подобного рода задачи – это ваша обычная работа. Если вы собираетесь заниматься решением конкретных проблем бизнеса, то с такими задачами вам сталкиваться вряд ли придется, и основная работа будет посвящена подготовке и интерпретации данных. Подготовке данных было уже уделено много внимания, поэтому сейчас рассмотрим исследовательскую задачу. Однако прежде чем перейти непосредственно к ее реализации, рассмотрим применение матричной записи данных в ней.

Так, если вы хотите взвесить некоторые значения X по некоторым весам W и получить сумму, то не обязательно писать выражение $w_1 \times x_1 + w_2 \times x_2$. Ту же самую операцию можно реализовать в виде матричного умножения. Первая матрица A будет представлять из себя X, где столбцы – признаки, а строки – традиционно объекты (если вы обрабатываете один объект за раз, то это может быть матрица с одной строкой). Вторая матрица B будет представлять из себя матрицу весов, где каждая колонка будет отождествляться с нейроном следующего слоя, а каждая строка будет отвечать за конкретный вес (значение веса). Однако даже матрица с одной строкой

в Python будет представлена как двумерный массив. Вы можете поэкспериментировать самостоятельно, выводя различные массивы\матрицы в консоль. После перемножения таких двух матриц получим новую матрицу C, где каждая строка будет представлять собой взвешенный набор признаков $X\ (x_1, \ldots, x_n)$ по $W\ (w_1, \ldots, w_n)$. Схематично это выглядит так, как показано на рисунке 65 [1].

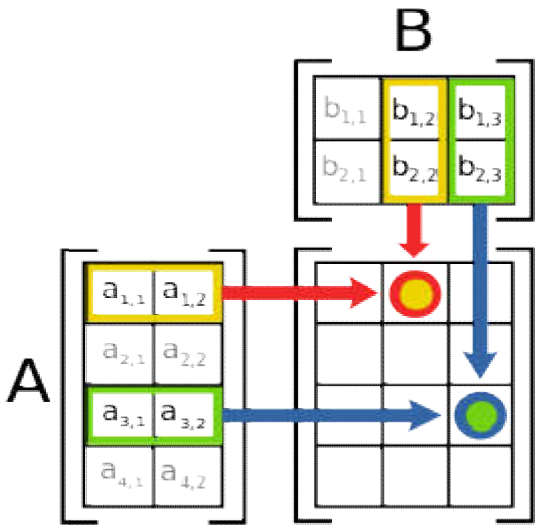


Рисунок 65. Умножение матриц

Необходимо отметить, что взвешивание значений с точки зрения матрицы происходит так, что из матрицы A берутся строки, а из матрицы B берутся колонки. Поэтому иногда массивы\матрицы нужно будет транспонировать. Напомним суть операции транспонирования, приведя в качестве примера рисунок 66.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 \\ 2 & 4 \end{bmatrix} \quad \text{и} \quad \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \end{bmatrix}$$

Рисунок 66. Транспонирование матриц

Объявления функций активации и ее производной представлены в листинге 59.

Листинг 59

```
# -*- coding: utf-8 -*-
#Необходимый импорт
import numpy as np
#Определяем функцию активации
def activation(x):
    return 1 / (1 + np.exp(-x))
#Определяем производную от функции активации
def sigma_derivative(x):
    return x * (1 - x)
```

Реализация алгоритма обраного распространения ошибки с подробными комментариями приведена в листинге 60. Забегая вперед, скажем, что в практической части вам предлагается задача модификации данного кода и интерпретации его результатов.

Листинг 60

```
# X - это матрица, где столбцы это признаки,
# а строки это объекты выборки.
X = np.array([[0, 0, 1],
              [0, 1, 0],
              [1, 0, 0],
              [1, 1, 1]])
# Y - это матрица, где столбцы это признаки
# (в данном случае один целевой признак),
# а строки это объекты выборки.
y = np.array([[0],
              [1],
              [1],
              [0]])
# Зафиксируем генератор случайных чисел, чтобы получать каждый раз
#предсказуемый (одинаковый) результат
np.random.seed(4)
# Нотация слоев - L1, L2, L3.
# В каждом слое есть n1, n2, n3 нейронов.
# Строки в матрицах индексируются как i, столбцы как j.
# W_1_2 - это матрица весов между первым и вторым слоем.
# Строки определяют левый нейрон (т.е. нейрон первого слоя) и соответственно
количество строк равно n1.
```

Продолжение листинга 60

```
# Столбцы определяют правый нейрон (т.е. нейрон второго слоя) и соответственно количество столбцов равно n2.
# На пересечении i-й строки и j-го столбца получаем ячейку с конкретным весом, который
# связывает i-ый левый нейрон (слоя L1) и j-й правый нейрон (слоя L2)
W_1_2 = 2 * np.random.random((3, 2)) - 1
# W_2_3 - это матрица весов между вторым и третьим слоем. Все остальное как в W_1_2
W_2_3 = 2 * np.random.random((2, 1)) - 1
#скорость движения по антиградиенту
speed = 1.1
#цикл прогона модели
for j in range(100000):
    # lo, l1, l2 - матрицы определенного слоя сети. Каждая строка матрицы это реакция #на i-й объект входа. Каждая колонка матрицы это реакция j-го нейрона #соответствующего слоя на разные входные образы.
    # На пересечении i-й строки и j-го столбца получаем ячейку с конкретной #реакцией j-го нейрона на конкретный вход.
    # Первый слой нейронов полностью принимает значения входа X (т.е. все реакции #единичные).
    l1 = X
    # Второй слой нейронов рассчитывается как функция активации по каждому #элементу матрицы U
    # По сути l2 это матрица 4 на 4, где в каждой ячейке результат активации нейрона #второго слоя для всех 4-х входных образов
    # Детально:
    # матрица U = np.dot(l0, W_0_1) является результатом
    # матричного произведения выходов нейронов предыдущего слоя на веса между 1 #и 2 слоем.
    # Строки матрицы U отвечают за конкретный входной образ (объект).
    # Столбцы матрицы U отвечают за нейроны правого слоя (L2).
    # На пересечении i-й строки и j-го столбца получаем ячейку с конкретной #взвешенной суммой
    # для i-го входного образа и j-го нейрона слоя (l2). Иными словами, для каждого #нейрона слоя L2 и для каждого входа
    # считается U = W1X1 + W2X2 + W3X3 (поскольку входной нейрон содержит 3 #нейрона).
    l2 = activation(np.dot(l1, W_1_2))
    # тоже самое проделываем для Третьего слоя
```

Продолжение листинга 60

```
    # По сути l3 это матрица 4 на 1, где в каждой ячейке результат активации нейрона
#третьего слоя (а он один)
    #  для всех 4-х входных образов
    l3 = activation(np.dot(l2, W_2_3))
    # рассчитывает ошибку на выходе
    l3_error = y - l3
# рассчитывает модуль средней ошибки
    if (j % 10000) == 0:
        print "Error:" + str(np.mean(np.abs(l3_error)))
    # sigma - есть локальный градиент ошибки
    # l3_sigma рассчитывается как ошибка выхода всей сети на производную функции
#активации всех нейронов L3.
    # На пересечении i-й строки и j-го столбца получаем ячейку с конкретной сигмой
    # для i-го входного образа и j-го нейрона слоя (l3). Иными словами, для каждого
#нейрона слоя L3
    # и для каждого входа рассчитывается производная по функции активации
#умноженная на ошибку.
    # То есть l3_sigma будет матрица 4 на 1 (поскольку в L3 только один нейрон).
    l3_sigma = l3_error * sigma_derivative(l3)
    print l3_sigma
    # Ошибка L2 слоя оценивается через взвешенную сигму слоя L3 по весам между
#L2 и L3
    # Поскольку l3_sigma это матрица 4 на 1 и матрица W_2_3 4 на 1, то последнюю
#матрицу
    # надо Транспонировать, чтобы выполнить правило умножения матриц и взвесить
#элементы l3_sigma
    # по элементам W_2_3.
  # Тогда итоговая матрица l1_error будет 4 на 4, где на пересечении i-й строки и j-го
#столбца
    # будет ячейка с конкретной ошибкой j-го нейрона слоя L2 для i-го входного
#образа.
    l2_error = l3_sigma.dot(W_2_3.T)
    print l2_error
    # l2_sigma рассчитывается как ошибка слоя L2 на производную функции активации
#всех нейронов L2. На пересечении i-й строки и j-го столбца получаем ячейку с
#конкретной сигмой для i-го входного образа и j-го нейрона слоя (L2). Иными
#словами, для каждого нейрона слоя L2 и для каждого входа рассчитывается
#производная по функции активации, умноженная на ошибку.
```

Окончание листинга 60

```
    # То есть l2_sigma будет матрица 4 на 4 (поскольку в L2 четыре нейрона).
    l2_sigma = l2_error * sigma_derivative(l2)
    print l2_sigma
   # обновляем веса
    # l2 это матрица 4 на 4, где в каждой ячейке результат активации нейрона второго
#слоя для всех 4-х входных образов (по строкам).
    # A l3_sigma это матрица 4 на 1, где в каждой строке локальный градиент ошибки
#для 4-х входных образов.
    # Чтобы взвесить столбец матрицы l2 по локальному градиенту (l3_sigma), нужно
#транспонировать матрицу l2, чтобы
    # результат активации каждого нейрона в слое L2 по каждому входному образу
#вытянулся в строку.
    # Тогда соответствующая строка умножится на столбик l3_sigma.
    # Заметьте, что транспонирование l3_sigma не даст результата, так как тогда в
#каждом столбце
    # l3_sigma будет по одному значению, а в каждой строке l2 по 4 значения, а значит
#такое перемножение матриц не пройдет. Применяем транспонирование к l2.
    W_2_3 += speed * l2.T.dot(l3_sigma)
    # аналогично W_2_3
    W_1_2 += speed * l1.T.dot(l2_sigma)
# Прямое распространение для тестовых данных
X_test = np.array([[0, 0, 0],
                   [0, 1, 1],
                   [1, 0, 1],
                   [1, 1, 0],
                   [0.5, 0.5, 0],
                   [0.5, 0.5, 1]])
# Y_test должен получиться [1, 0, 0, 1, 1, 0]
l1 = X_test
l2 = activation(np.dot(l1, W_1_2))
l3 = activation(np.dot(l2, W_2_3))
print l3
```

Регуляризация и сеть прямого распространения

В отличие от предыдущих пунктов здесь не будет кода целиком, а только общие рекомендации по выполнению экспериментов, позволяющих выявить влияние регуляризации на многослойную сеть прямого распространения. В ходе работы над пунктом вы научитесь

генерировать синтетические данные, проводить серии экспериментов, анализировать возможности моделей на разнотипных данных и сравнивать результаты экспериментов.

Необходимые для выполнения работы операторы import представлены в листинге 61.

Листинг 61

```
import numpy as np
from matplotlib import pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.datasets import make_moons, make_circles, make_classification
from sklearn.neural_network import MLPClassifier
```

Первый шаг, который вам необходимо выполнить, – генерация данных. Библиотека Scikit-learn содержит множество наборов данных (sklearn.datasets). Среди этого набора есть реальные данные (выборки, которые можно загрузить при помощи функций – Loaders), так и специальные генераторы синтетических данных – Samplesgenerator. Подобные генераторы позволяют создать данные на лету по заданными распределениям, функциям, с возможностью добавления шума, случайных отклонений, смещений и т. п.

Есть несколько классических сценариев для проверки возможностей моделей в машинном обучении. Мы воспользуемся одним генератором линейных задач (datasets.make_classification) и двумя нелинейными генераторами (make_moons, make_circles). Отметим, что одной из важнейших характеристик данных является размерность. Важно понимать, что величина размерности данных соответствует количеству признаков, которые описывают исходные объекты. То есть каждая ось декартового пространства закрепляется за одним количественным признаком. Не все генераторы данных позволяют задавать размерность данных. Тем ни менее ниже приведены крайне важные распределения данных, которые иллюстрируют разные типы задач в машинном обучении. Соответственно тестирование моделей

на подобных данных позволяет без наличия реальных данных проверять какие-либо гипотезы.

Пример генераторов:

- make_classification – позволяет сгенерировать такие классы-признаки, которые будут линейно разделяться в n-мерном пространстве (см. рисунок 67. а).
- make_circles – позволяет сгенерировать такие классы-признаки, что признаки одного класса геометрически окружают признаки другого класса (см. рисунок 67.б).
- make_moons – позволяет сгенерировать такие классы-признаки, что признаки одного класса частично пересекаются с признаками другого класса (см. рисунок 67. в).

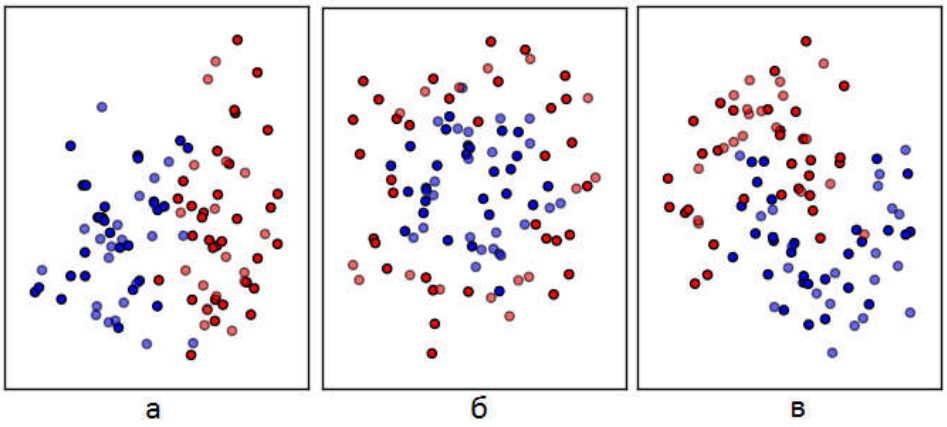


Рисунок 67. Виды распределений признаков, сгенерированных инструментами Scikit-learn

Самостоятельно найдите в Scikit информацию по различным генераторам и сгенерируйте синтетические данные для проведения экспериментов. Пример кода для создания данных трех разнотипных задач классификации приведен в листинге 62.

Для удобства дальнейшей работы рекомендуется объединить все наборы данных в список, как показано в листинге 63, тогда вы сможете организовать цикл по каждому элементу этого списка (что приведено в коде). В этом же листинге есть пример масштабирования данных и разбиения их на тренировочную тестовую выборку.

Листинг 62

```
X, y = make_classification(n_features=2, n_redundant=0, n_informative=2,
random_state=0, n_clusters_per_class=1)
rng = np.random.RandomState(2)
X += 2 * rng.uniform(size=X.shape)
linearly_dataset = (X, y)
moon_dataset = make_moons(noise=0.3, random_state=0)
circles_dataset = make_circles(noise=0.2, factor=0.5, random_state=1)
```

Листинг 63

```
datasets = [moon_dataset, circles_dataset, linearly_dataset]
for ds_cnt, ds in enumerate(datasets):
    X, y = ds
    X = StandardScaler().fit_transform(X)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4, random_state=42)
```

Функция train_test_split делит данные так, что тестовая выборка составляет 40% от исходного набора данных. Разделение происходит случайным образом (т.е. элементы берутся из исходной выборки не последовательно). В этом же цикле можно организовать обучение моделей, их тестирование и формирования графиков.

В теле цикла создайте коллекцию многослойных сетей (MLPClassifier) с различными коэффициентами регуляризации (alpha) и обучите каждую модель на каждом типе данных. Всего должно получиться 15 моделей (3 типа данных\задачи и 5 видов коэффициентов регуляризации). Для реализации используйте код из листинга 64.

Листинг 64

```
alphas = np.logspace(-5, 3, 5)
```

Следующим шагом отобразите исходные данные и результаты на графиках. Пример отображения исходных данных уже был приведен в предыдущих пунктах. Однако на этот раз необходимо построить не один график, а серию графиков в одном окне. В итоге должна получиться таблица из графиков (3 строки, 6 столбцов). Для этого изучите функцию subplot из библиотеки matplotlib. Пример кода для инициализации нового графика (если точнее, то подграфика в рамках одного полотна\окна) показан в листинге 65.

Листинг 65

```
current_subplot= plt.subplot(3, 5 + 1, i)
```

Показанным в листинге 65 способом мы можем получить один из графиков в полотне и заполнять его данными, а потом таким же образом получить следующий график (то есть объект current_subplot). В первую очередь на всех графиках необходимо отобразить исходные данные (точки\признаки). Можете отобразить тренировочную и тестовую выборку или только одну. Пример кода, в котором точки тестовой выборки рисуются полупрозрачными (alpha=0.6), приведен в листинге 66.

Листинг 66

```
current_subplot.scatter(X_train[:, 0], X_train[:, 1], c=y_train, cmap=cm_bright)
current_subplot.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,alpha=0.6)
```

В листинге 67 cm и cm_bright это цветовые схемы для отрисовки графиков. Их можно взять как из готового набора, так и задать вручную. Это делается следующим образом (листинг 65):

Листинг 67

```
cm = plt.cm.RdBu
cm_bright = ListedColormap(['#FF0000', '#0000FF'])
```

При отрисовке графиков рекомендуется придерживаться следующего. Пусть левый столбец графиков отображает только исходные данные и ничего больше. Остальные же столбцы должны отображать функции обученных моделей в пространстве исходных данных. Это необходимо, чтобы визуально оценить, как каждая модель разделила исходное пространство признаков. Вместо построения значений функции Y на отдельной оси отобразите значения Y в виде цветовых градиентов (чтобы не строить трехмерные графики). Пример визуализации показан на рисунке 68.

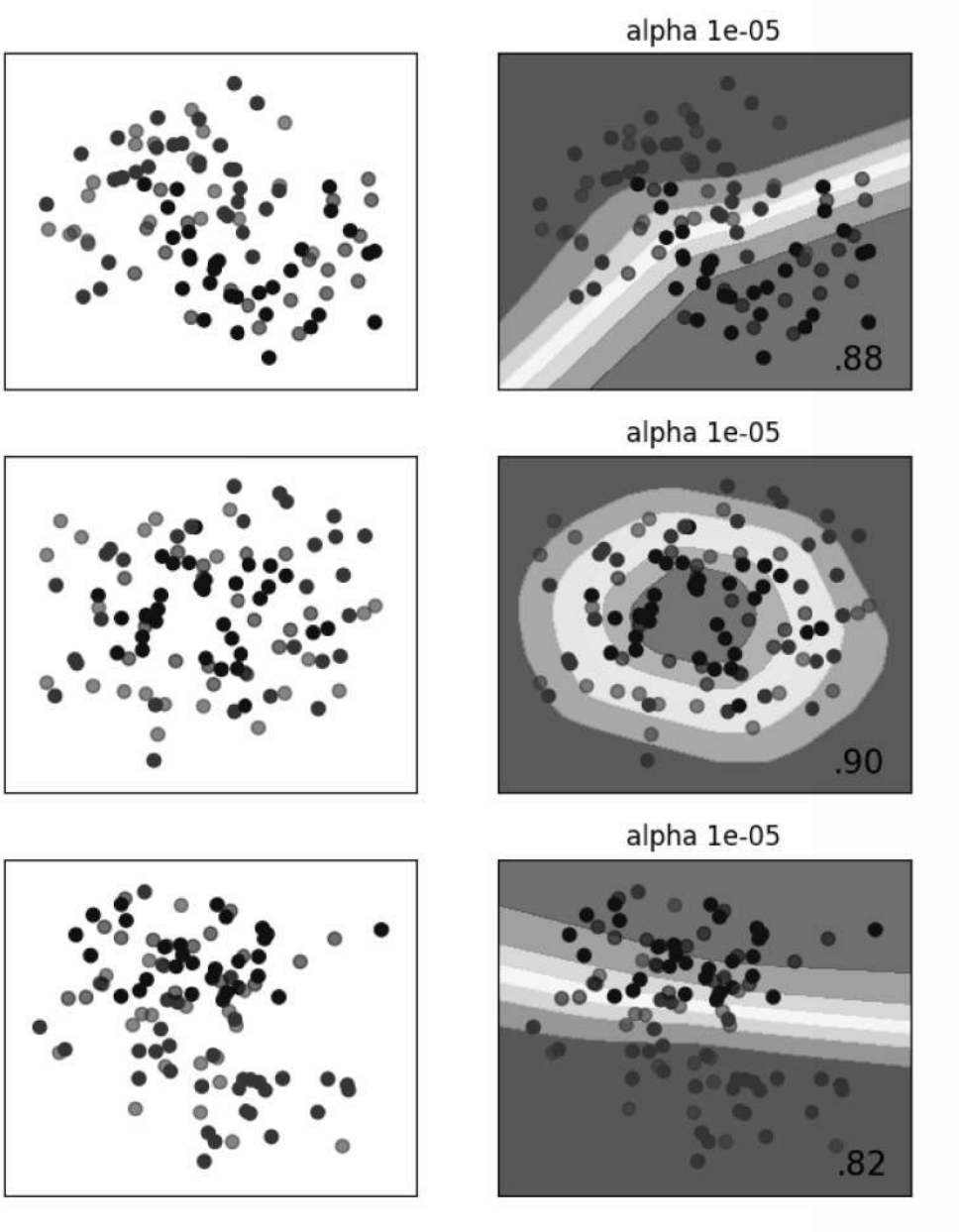


Рисунок 68. Пример визуализации качества моделей

Здесь слева представлена колонка с тремя графиками исходных распределений признаков (на каждом графике две группы точек исходных данных). Каждая строка отвечает за свою задачу классификации (moondataset, circledataset, linearseparabledataset). Правая колонка демонстрирует как конкретная модель многослойной сети MLP (с параметром alpha=0.00001) классифицирует исходное распределение. Значения модели в каждой точке признаков представлены цветовым градиентом. Это гораздо более удобный способ визуализации, чем построение трехмерных поверхностей.

Как мы видим, правый столбец отображает градиент. Чтобы получить градиент, необходимо определить множество точек, по которым будет строиться график. Это можно сделать при помощи регулярной сетки, как показано в листинге 68.

Листинг 68

```
h = .02  #шаг регулярной сетки
x0_min, x0_max = X[:, 0].min() - .5, X[:, 0].max() + .5
x1_min, x1_max = X[:, 1].min() - .5, X[:, 1].max() + .5
xx0, xx1 = np.meshgrid(np.arange(x0_min, x0_max, h), np.arange(x1_min, x1_max, h))
```

В листинге 69 представлено несколько вариантов кода для получения значений функции модели в указанных точках сетки.

Листинг 69

```
Z = clf.decision_function(np.c_[xx0.ravel(), xx1.ravel()])#1
Z = clf.predict_proba(np.c_[xx0.ravel(), xx1.ravel()])[:, 1]#2
Z = clf.predict(np.c_[xx0.ravel(),xx1.ravel()])#3
```

Сделаем некоторые пояснения к коду:

1. decision_function возвращает расстояние до разделяющей гиперплоскости в соответствующем измерении (проекционной оси).
2. predict_proba возвращает вероятность принадлежности к каждому классу (так для двух классов можно брать одну из вероятностей (один из столбцов двумерного массива) и исходя из значения раскрашивать область в соответствующий цвет).
3. Predict возвращается целочисленную оценку принадлежности к классу (как бинарная маска).

Третий вариант в данном случае выдаст округленные значения классов (0, 1 и т. д.), а функция predict_proba выдает сглаженную функцию вероятности класса. Чтобы понять разницу, попробуйте различные варианты.

Однако не все классификаторы имеют decision_function. Чтобы узнать, содержит ли текущий объект интересующую нас функцию, можно воспользоваться методом hasattr. Этот метод возвращает флаг,

указывающий на то, содержит ли объект указанный атрибут и нужен для обработки разных вариантов классификаторов (можно выстроить на этом if-else логику). Пример ее вызова показан в листинге 70.

Листинг 70

```
hasattr(clf, "decision_function")
```

Далее вам необходимо нарисовать результат, оси, заголовок и текст со значением точности модели. Выполнить это можно с помощью кода из листинга 71.

Листинг 71

```
Z = Z.reshape(xx0.shape)
current_subplot.contourf(xx0, xx1, Z, cmap=cm, alpha=.8)
current_subplot.set_xlim(xx0.min(),xx0.max())
current_subplot.set_ylim(xx0.min(),xx1.max())
current_subplot.set_xticks(())
current_subplot.set_yticks(())
current_subplot.set_title(name)
current_subplot.text(xx0.max() - .3, xx1.min() + .3, ('%.2f' % score).lstrip('0'),
size=15, horizontalalignment='right')
```

После отрисовки градиента добавьте на каждый график тестовую выборку (точки из каждой задачи для графика каждой модели). Это можно сделать таким же образом, как показано в листинге 72.

Листинг 72

```
current_subplot.scatter(X_test[:, 0], X_test[:, 1], c=y_test, cmap=cm_bright,alpha=0.6)
```

Не забудьте, что после построения каждого графика необходимо инкрементировать переменную i для функции subplot(), а также помните о необходимости применить функцию plt.show() в конце всех операций формирования графиков для вывода результата в окне ОС.

При желании вы можете настроить отступы графиков в окне просмотра при помощи следующего кода (листинг 73).

Листинг 73

```
figure.subplots_adjust(left=.02, right=.98)
```

Для задания заголовков и вывода текста на графиков можно использовать следующие функции из листинга 74.

Листинг 74

```
current_subplot.set_title("Input data")
current_subplot.text(xx0.max() - .3, xx1.min() + .3, ('%.2f' % score).lstrip('0'), size=15,
horizontalalignment='right')
```

Работа с библиотеками Keras и Theano. Настройка под Windows

Мы закончили рассмотрение простых моделей нейронных сетей и переходим к более сложным. В частности, к сверточным сетям. Однако поскольку в библиотеке scikit нет полноценной поддержки таких сетей, то в данном пункте мы рассмотрим установку и настройку библиотек Keras и Theano, потому что для пользователей Windows установка этих библиотек нетривиальна.

Для разворачивания Keras и Theano для CPU на Windows выполните следующее:

1. Откройте Anaconda prompt (командную строку Анаконды через меню «Пуск» или просто выполните стандартную команду cmd. Если все установлено верно, то Anaconda «перехватит» адресованные ей команды)
2. Выполните conda update conda
3. Выполните conda update --all
4. Выполните conda install mingw libpython
5. УстановитеTheano, pip install Theano
6. Установите Keras pip install keras
7. Выполните код для проверки корректности всех действий (он показан в листинге 75).

Листинг 75

```
import numpy as np
import sklearn.linear_model as lin
from keras.layers.core import Dense, Activation
from keras.models import Sequential
from keras.optimizers import RMSprop

x_train = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
y_train = np.array([0, 1, 1, 0]).reshape(4, 1)
x_test = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])

model = Sequential()
model.add(Dense(2, init='lecun_uniform', input_shape=(2,)))
model.add(Activation('relu'))
model.add(Dense(1, init='lecun_uniform'))
model.add(Activation('linear'))
rms = RMSprop(lr=0.01)
model.compile(loss='mse', optimizer=rms)

epochs = 200
model.fit(x_train, y_train, batch_size=1, nb_epoch=epochs, verbose=0)
y_predict = model.predict(x_test, batch_size=1)
print y_predict
```

Теперь опишем разворачивание Theano для GPU на Windows. Ведь обучение даже относительно простых сверточных сетей – крайне затратный вычислительный процесс, который может занять несколько часов на довольно мощном процессоре (здесь также стоит учитывать, что не все библиотеки и платформы поддерживают параллельное исполнение кода между ядрами процессоров, а значит, попросту не используют всю вычислительную мощь CPU).

Современные графические карты позволяют запускать очень много параллельных вычислений на аппаратном (а не только на программном) уровне. Лидером таких вычислений на сегодняшний день является компания NVidia, которая предлагает специальную платформу под названием CUDA. Это программно-аппаратная платформа, которая встроена в видеокарты и драйвера NVidia. Подобная

технология может ускорить вычисления во много раз (от 4 до 100). Конечно же, GPU вариант предпочтительнее не только в реальных проектах и промышленных системах, но и в процессе обучения, так как может существенно сэкономить время на эксперименты и выполнение учебных заданий.

Библиотека Theano может исполнять свои модели на графическом процессоре (GPU), который установлен в системе. Однако поддерживаются только графические процессоры от NVidia и только те, которые поддерживают режим CUDAComputeCapability 3.0 или выше. Узнать о том, какой режим поддерживает ваша видеокарта можно на официальном сайте NVidia.

Само разворачивание Theano состоит из следующих шагов:

1. Проверьте, что у вас есть свежий драйвер на видеокарту Nvidia. Необходима именно карта NvidiacCompute Capablity = 3.0 или выше (проверить Compute Capablity своей видеокарты можно тут [58]). Если карты от Nvidia нет, то этот сценарий реализовать не получиться и просто используйте CPU.
2. Скачайте и установите CUDA Toolkit 8.0 отсюда: [59].
3. Скачайте Cudnn 5.0 для CUDA Toolkit 8.0 отсюда [60]. Это библиотека для низкоуровневой поддержки нейронных сетей. Распакуйте архив Cudnn. Там будет несколько папок. В каждой папке по одному файлу. Эти файлы нужно переместить в место, куда был установлен CUDA Toolkit 8.0. Например, файл cudnn64_5.dll из папки cuda\bin положить в папку bin в место, куда установился CUDA Toolkit 8.0 и так далее.
4. Теперь код Theano может запускаться на GPU. Но его еще нужно скомпилировать. Для компиляции нужен VC++ компилятор от Майкрософта. Нужную версию компилятора можно получить, установив Visual Studio 2012.
5. Создайте текстовый файл с именем .theanorc, заполненный как показано в листинге 76. Описание приведено в листинге 77.

6. Созданный файл нужно положить в домашнюю или пользовательскую директорию ОС. Для Win это C:\Users\$UserName$. Чтобы четко определить директорию, можно выполнить следующий код (листинг 78).

Листинг 76

```
[global]
floatX=float32
device=gpu
mode=FAST_RUN
cnmem=0.7
 [cuda]
root=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v7.5
 [nvcc]
flags=-LD:\Program_Files\Anaconda\libs
–compiler_bindir=C:\Program_Files_(x86)\Microsoft_Visual_Studio_11.0\VC\bin\
–fastmath=True
```

Листинг 77

```
[global]
Cnmem - объем памяти, который используется на видео карте
 [cuda]
Root - путь на корневую папку CUDAtoolkit. Например:
«root=C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v8.0»
 [nvcc]
Flags - указывает путь до библиотек Anaconda
compiler_bindir - путь до компилятора cl.exe из VisualStudio
Во всех параметрах [nvcc] не должно быть пробелов. Так «Program Files (x86)» нужно заменить на  «Program_Files_(x86)». Пример пути:
«D:\Program_Files\Visual_Studio_1\VC\bin\»
```

Листинг 78

```
os.environ['THEANO_FLAGS']="device=gpu0,lib.cnmem=0.5"
path=os.path.expanduser('~/.theanorc.txt')
```

7. Выполните проверочный код из листинга 79. Он должен вывести 'Used the gpu' и нулевой код ошибки.

Листинг 79

```
from theano import function, config, shared, sandbox
import theano.tensor as T
import numpy
import time
 vlen = 10 * 30 * 768  # 10 x #cores x # threads per core
iters = 10000
  rng = numpy.random.RandomState(22)
x = shared(numpy.asarray(rng.rand(vlen), config.floatX))
f = function([], T.exp(x))
print(f.maker.fgraph.toposort())
t0 = time.time()
for i in range(iters):
    r = f()
t1 = time.time()
print("Looping %d times took %f seconds" % (iters, t1 - t0))
print("Result is %s" % (r,))
if numpy.any([isinstance(x.op, T.Elemwise) for x in f.maker.fgraph.toposort()]):
    print('Used the cpu')
else:
    print('Used the gpu')
```

Получение данных средствами Keras

Подобно библиотеке Scikit-learn библиотека Keras также имеет встроенные функции для получения готового набора данных. Пример показан в листинге 80. На выходе у нас получаются стандартные массивы Numpy.

Листинг 80

```
from keras.datasets import mnist
(X_train, y_train), (X_test, y_test) = mnist.load_data()
```

Забежав немного вперед, скажем, что сверточные модели, представленные библиотекой Keras, принимают на вход немного непривычный формат данных для обучения. Так параметр Input_shape функции model.fit() имеет следующую форму: (samples, channels, rows, cols). Здесь samples – количество элементов в выборке (может

не указываться), channels – количество каналов (если речь идет об RGB-изображениях, то на каждую компоненту будет по матрице и этот параметр будет равен трем), rows и cols соответственно ширина и высота матрицы (изображения).

Исходя из этого, нам необходимо модифицировать полученные данные. Чтобы сменить форму массива Numpy, выданного load_data, воспользуйтесь функцией reshape, которая в качестве параметров принимает размер каждого измерения. Пример функции показан в листинге 81.

Листинг 81

```
X_train = X_train.Reshape(N, 1, r, c)
```

Кроме смены формы массива его элементы необходимо привести в диапазон от 0 до 1 для корректной работы с ним сверточной нейронной сетью. Сделайте это самостоятельно.

Над выборкой Y также необходимо провести преобразования. Целое число нужно перевести в бинарную маску (вектор) так, чтобы класс 2 из трех возможных классов представлял собой вектор [0,0,1] при отсчете от нуля. Подобное преобразование можно сделать при помощи функции np_utils.to_categorical.

Создание и обучение модели сверточной сети

Общая логика обучения модели такая же, как и в моделях библиотеки Scikit-Learn. Но поскольку Keras больше ориентирована на нейросетевые модели, то она позволяет более детально настраивать структуру и работу сети. В этом пункте мы рассмотрим лишь некоторые из возможностей.

Модели в Keras описываются как вычислительные графы. Поэтому, чтобы создать и обучить сверточную сеть, сначала необходимо создать последовательную модель (граф последовательных вычислений\обработчиков, «контейнер моделей»). Сделать это можно так, как показано в листинге 82.

Листинг 82

```
model = Sequential()
```

После создания граф необходимо заполнить рядом обработчиков. Описание всех возможных обработчиков есть в документации Keras, например, здесь: [61]. Ниже представлен список основных обработчиков, с которыми мы столкнемся в этом пункте.

Dense – одномерный слой обычных нейронов. По сути, этот обработчик вычисляет взвешенные суммы для каждого нейрона слоя, но не вычисляет для них функцию активации (так как она задается отдельным обработчиком). Функция, создающая обработчик, принимает на вход число нейронов. Использовать ее можно так, как показано в листинге 83.

Листинг 83

```
Dense(128)
```

ConvolutionND – сверточный слой нейронов. Данный обработчик подсчитывает взвешенные суммы для каждой карты и каждого нейрона в слое, но также не вычисляет для них функцию активации. Функция, создающая обработчик принимает на вход число карт в слое и размер ядра\фильтра, также определяется форма входа и режим обработки границ (valid означает, что все ядра поместятся в исходную матрицу и не будет выступающих границ, при этом карта будет меньше исходной матрицы; в случае same карта будет такого же размера, как и входная матрица). Можно задать и ряд других параметров, но с ними вам предлагается ознакомиться самостоятельно. Пример использования приведен в листинге 84.

Activation – обработчик функции активации. Задается после каждого слоя нейронов (Convolution или Dense), такие обработчики, как MaxPooling2D или Dropout, не являются слоями нейронов. Данный обработчик рассчитывает значение активации для взвешенных сумм предыдущего слоя. В качестве аргумента функция прини-

мает строку с названием собственно функции активации (relu, tanh, linear, sig, prelu). Пример использования – в листинге 85.

Листинг 84

```
Convolution2D(nb_filters, nb_conv, nb_conv,
border_mode='valid',
input_shape=(1, img_rows, img_cols)))
```

Листинг 85

```
Activation('relu')
```

MaxPoolingND – обработчик объединения по максимуму. Обработчик проходит по матрице, поступающей на вход, окном размера pool_size (например, 2X2) и подает на выходную матрицу одно максимальное значение из данной области входной матрицы. Иллюстрация работы функции – рисунок 69 [1].

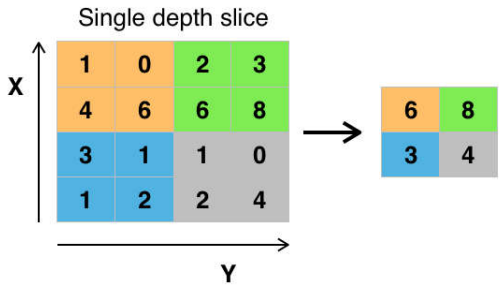


Рисунок 69. Иллюстрация действия обработчика MaxPooling2D

Как мы видим, из входной матрицы 4X4 формируется матрица 2X2 максимальных значений. Пример вызова функции приведен в листинге 86.

Листинг 86

```
MaxPooling2D(pool_size=(nb_pool, nb_pool))
```

Dropout – данный обработчик задает степень разрыва нейронных связей между соседними слоями. В качестве параметра функция принимает значение от 0 до 1, выражающее процент разрыва. В данном случае 25% связей между слоями выродятся в ноль и не будут меняться в процессе обучения.

Обратите внимание!

- *Такой возможности не было в библиотеке scikit-learn. Это важная настройка, так как она позволяет контролировать переобучение сети не только регуляризацией. Кроме того, таким способом можно добиться эффекта локальности.*

Вызов обработчика – в листинге 87.

Листинг 87

```
Dropout(0.25)
```

Flatten – выравнивающий обработчик. Преобразует связи от нейронов слоя свертки к одномерному слою обычных нейронов. Такой обработчик используется ближе к завершению сети, чтобы формировать выход модели. Пример вызова – в листинге 88.

Листинг 88

```
Flatten()
```

Если создать модель из двух сверточных слоев с функцией активации «relu», затем перейти к обычным слоям, добавить разрывы и завершите модель слоем из 10 нейронов с функцией активации «softmax», то данная функция сформирует выходной вектор таким образом, чтобы сумма всех элементов была равна единице, а отдельный элемент вектора выражал вероятность принадлежности входного образа к классу, который отображается этим элементом вектора.

Чтобы добавить тот или иной обработчик в вычислительный граф, используется функция add, как показано в листинге 89.

После формирования графа необходимо скомпилировать модель. Причем то же самое придется выполнить и после загрузки модели из файла, поскольку сохраняется лишь граф вычислений, а не готовый к исполнению объект. Пример компиляции модели приведен в листинге 90.

Листинг 89

```
model.add(<обработчик>)
```

Листинг 90

```
model.compile(loss='categorical_crossentropy',
optimizer='adadelta',
metrics=['accuracy'])
```

Обучается модель уже знакомым способом (с помощью функции fit, куда передаются тренировочные выборки X, Y). Однако в данных моделях имеет смысл рассмотреть еще несколько параметров:

- batch_size – определяет количество предъявляемых образов в рамках одного обновления весов (так если batch_size>1, то производиться обучение по группе, а не по одиночным образам). Данный параметр может повлиять на скорость обучения (особенно на CPU) и на объем требуемой памяти. Так небольшие группы по 50 образов могут потребовать меньше памяти и вычислительных ресурсов (слово «могут» используется потому, что есть и другие факторы, в том числе аппаратная реализация некоторых функций и поведение кэша).
- nb_epoch – определяет количество эпох обучения.
- verbose – определяет, выводить ли в консоль детали обучения или нет. Под деталями понимается информация о ходе обучения, времени и оценке качества за каждую итерацию.
- validation_data – определяет набор данных, по которому будет проводиться оценка качества модели.

Пример обучения модели приведен в листинге 91.

Листинг 91

```
model.fit(X_train, Y_train, batch_size=batch_size, nb_epoch=nb_epoch,
verbose=1, validation_data=(X_test, Y_test))
```

Качество обученной модели проверяется кодом из листинга 92.

Листинг 92

```
score = model.evaluate(X_test, Y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Напомним, что:

- Величина loss характеризует интегральную оценку всех среднеквадратических ошибок. Единицы этой величины совпадают с единицами Y. Чем меньше это значение, тем лучше. Однако для каждой задачи (каждого типа данных) конкретные величины будут разными, и оценивать эту величину нужно\можно, только зная характер данных и допустимые величины. Так, для оценки стоимости квартиры общая ошибка 300 000 на тренировочной выборке в несколько тысяч квартир может быть вполне приемлемой. А вот такая же величина loss для оценки уровня лейкоцитов в крови на тысячу пациентов просто неприемлема.
- Величина accuracy характеризует процент верных ответов из тестовой выборки. Чем ближе это значение к 100% (или 1.00), тем лучше.

Загрузка и сохранение сложных моделей

В одном из прошлых пунктов был рассмотрен вариант сохранения объектов посредством функций pickle.dump(obj, output) и pickle.load(f). Однако многие модели Keras имеют такую сложную структуру, что стандартный сериализатор объектов вызывает исключение из-за глубины ссылок. Поэтому для сохранения и загрузки сложных моделей нейронных сетей необходимо использовать следующий код (листинг 93).

Обратите внимание!

- *Настройки модели и веса сохраняются раздельно.*

Листинг 93

```
def SaveToJSon(model, fileName):
    model_json = model.to_json()
with open(fileName + ".net", "w") as json_file:
        json_file.write(model_json)
    model.save_weights(fileName + ".wgt")
print("model saved to disk")
def LoadFromJSon(fileName):
    json_file = open(fileName + ".net", 'r')
    loaded_model_json = json_file.read()
    json_file.close()
    loaded_model = model_from_json(loaded_model_json)
    loaded_model.load_weights(fileName + ".wgt")
print("Loaded model from disk")
return loaded_model
```

Рекуррентные сети для прогнозирования временных рядов

Теперь рассмотрим пример работы с рекуррентными сетями средствами библиотеки Keras, а именно, их применение для решения задачи прогнозирования временных рядов. Пусть в файле «international-airline-passengers.csv» (скачать его можно отсюда: [41]) находятся данные о ежемесячном количестве пассажиров международного аэропорта в виде, представленном на рисунке 70. Применим рекуррентную сеть для прогнозирования количества пассажиро в будущем. Глубину прогноза возьмем равной одному месяцу, как и количество точек, по которому будем строить прогноз. Качество работы модели оценим по величине среднеквадратичной ошибки. Данная задача и код ее решения, показанный в листинге 94, взяты из [62].

Листинг 94

```
# -*- coding: utf-8 -*-
#Необходимые импорты
import numpy
import matplotlib.pyplot as plt
```

Продолжение листинга 94

```
from pandas import read_csv
import math
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
# функция для создания выборки из исходного массива данных
def create_dataset(dataset, look_back=1):
  dataX, dataY = [], []
  for i in range(len(dataset)-look_back-1):
    a = dataset[i:(i+look_back), 0]
    dataX.append(a)
    dataY.append(dataset[i + look_back, 0])
  return numpy.array(dataX), numpy.array(dataY)
# фиксация состояния рандома для получения предсказуемого результата
numpy.random.seed(7)
# загрузка данных
dataframe = read_csv('international-airline-passengers.csv', usecols=[1], engine='python',
skipfooter=3)
dataset = dataframe.values
dataset = dataset.astype('float32')
# нормализация данных
scaler = MinMaxScaler(feature_range=(0, 1))
dataset = scaler.fit_transform(dataset)
# ручное разделение данных на обучающую и тестовую выборку
train_size = int(len(dataset) * 0.67)
test_size = len(dataset) - train_size
train, test = dataset[0:train_size,:], dataset[train_size:len(dataset),:]
# Изменение исходного набора данных для того, чтобы в каждой строке X оказался
# временной ряд размером look_back, начиная от момента времени t0. A в Y
# временной ряд единичного размера от момента времени t0 + look_back.
# В данном примере, для look_back=4  X1 будет состоять из объектов:  Data[t0],
# Data[t1], Data[t2], Data[t4], а Y1 из объектов: Data[t5]. X2 начнется с элемента
# Data[t1] и т.д.
 trainX, trainY = create_dataset(train, look_back)
testX, testY = create_dataset(test, look_back)
# изменение формы входных данных и приведение их к виду
# [количество объектов, размер временного ряда из объектов, количество признаков
каждого объекта]
```

Окончание листинга 94

```
trainX = numpy.reshape(trainX, (trainX.shape[0], trainX.shape[1], 1))
testX = numpy.reshape(testX, (testX.shape[0], testX.shape[1], 1))
# Создание графа вычислений (некоторой абстрактной модели)
model = Sequential()
#4 – это количество блоков LSTM, input_shape - это форма вектора входа
model.add(LSTM(4, input_shape=(look_back, 1)))
model.add(Dense(1))
#Компиляция модели
model.compile(loss='mean_squared_error', optimizer='adam')
#Обучение модели
model.fit(trainX, trainY, epochs=100, batch_size=1, verbose=2)
# выполнение предсказания
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
# так как прогнозное значение нормализовано, то необходимо его восстановление
# под восстановлением понимается приведение к исходному масштабу данных
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
# вычисление ошибки прогноза
trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:,0]))
print('Train Score: %.2f RMSE' % (trainScore))
testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
print('Test Score: %.2f RMSE' % (testScore))
# подготовка данных для отрисовки
trainPredictPlot = numpy.empty_like(dataset)
trainPredictPlot[:, :] = numpy.nan
trainPredictPlot[look_back:len(trainPredict)+look_back, :] = trainPredict
testPredictPlot = numpy.empty_like(dataset)
testPredictPlot[:, :] = numpy.nan
testPredictPlot[len(trainPredict)+(look_back*2)+1:len(dataset)-1, :] = testPredict
# построение графика
plt.plot(scaler.inverse_transform(dataset))
plt.plot(trainPredictPlot)
plt.plot(testPredictPlot)
plt.show()
```

```
"Month","International airline passengers: monthly totals in thousands. Jan 49 ? Dec 60"
"1949-01",112
"1949-02",118
"1949-03",132
"1949-04",129
"1949-05",121
"1949-06",135
"1949-07",148
"1949-08",148
"1949-09",136
"1949-10",119
"1949-11",104
"1949-12",118
"1950-01",115
"1950-02",126
"1950-03",141
"1950-04",135
"1950-05",125
"1950-06",149
"1950-07",170
```

Рисунок 70. Данные о пассажирах

На этом мы заканчиваем рассмотрение инструментальных средств для разработки приложений сферы машинного обучения на языке Python. Как вы могли заметить, в пособии не были рассмотрены реализации генетических алгоритмов и методов нечеткой логики. Это связано с тем, что их реализация на языке Pyhton не требует каких-либо специализированных библиотек, поэтому эту часть практического материала вам предлагается освоить самостоятельно в рамках выполнения задач по этим темам (см. раздел «Практические задания»). В случае затруднения вы можете обратиться к следующим ресурсам: реализация генетического алгоритма на Python – [63] и материалы учебного пособия [64], где рассматривается реализация генетических алгоритмов и некоторых операций нечеткой логики на языке Java.

КОНТРОЛЬНЫЕ ВОПРОСЫ И ТЕСТОВЫЕ ЗАДАНИЯ

Тест «Общие сведения о машинном обучении»

1. Выберите верные утверждения
   a) Одна из задач машинного обучения – научиться делать прогнозы для признаков
   b) Объекты описываются с помощью признаков
   c) Одна из задач машинного обучения – научиться делать прогнозы для объектов
   d) Признаки описываются с помощью объектов
2. Какие из этих задач являются задачами классификации?
   a) Прогноз температуры на следующий день
   b) Разделение книг, хранящихся в электронной библиотеке, на научные и художественные
   c) Поиск групп похожих пользователей интернет-магазина
   d) Прогноз оценки студента по пятибалльной шкале на экзамене по машинному обучению в следующей сессии
3. Какие свойства данных препятствуют однозначному построению разделяющей поверхности?
   a) Ортогональность
   b) Мультиколлинеарность
   c) Противоречивость
   d) Категориальность
4. Какая способность людей и систем позволяет получать им новые знания по наблюдению отдельных прецедентов (примеров)?
   a) Корректировать ошибку
   b) Обобщать
   c) Запоминать
   d) Распознавать образы
5. Какая задача лучше всего подходит под следующее описание. Нахождение такой функции F, которая бы наилучшим образом отображала неизвестные ранее объекты X в конечное множество

целочисленных номеров (имен, меток), на основании обучающих пар (X, Y)?

a) Прогнозирование денежных затрат
b) Кластеризация клиентов
c) Классификация образов
d) Выявление особенностей в данных

6. Почему для обучения моделей используются такие методы, как Градиентный спуск?
   a) Потому что метод позволяет корректировать параметры модели постепенно
   b) Потому что аналитические решения не всегда дают корректное решение
   c) Потому что такой подход позволяет получать более точные решения (Глобальный экстремум в отличие от локального)
   d) Потому что при большой размерности входных данных подобные методы работают быстрее
7. Выберите верные утверждения
   a) Метод Байеса – это во многом классический подход к классификации, основанный на оценке частоты встреч объектов со схожими признаками
   b) Благодаря универсальности статистического подхода метод Байеса позволяет решать любые задачи без априорной информации
   c) Данный метод позволяет очень хорошо обобщать высокоуровневые признаки
   d) Закон, задающий распределение вероятностей, который используется в предсказательной модели, сильно влияет на способ обобщения

Проблема переобучения

1. Какие факторы влияют на переобучение?
2. Какие есть способы оценки переобучения?
3. Какие есть способы борьбы с переобучением?

Регрессия

1. Почему такая простая формула, как y=kx+b, позволяет делать прогнозы или классификацию?
2. В чем отличие линейной и логистической регрессий?
3. В чем отличие линейной от нелинейной регрессии?
4. В чем отличие линейной регрессии от полиномиальной?
5. Что позволяет делать LASSO?
6. В чем заключаются особенности Ridge регрессии?

Модели и методы нечеткой логики

1. Охарактеризуйте следующие понятия: нечеткие множества, операции нечеткой логики, нечеткие модели или нечеткие системы.
2. Сформулируйте понятие лингвистической неопределенности.
3. Напишите функцию лингвистической неопределенности некоторого объекта.
4. Дайте определение функции принадлежности.
5. К какому типу относятся нечеткие множества, если значения функции принадлежности нечеткого множества представлены точными числовыми значениями?
6. К какому типу относятся нечеткие множества, если значения функции принадлежности нечеткого множества моделируются другими нечеткими множествами?
7. Какие существуют способы задания функции принадлежности?
8. Приведите три примера функции принадлежности, задаваемых функциональным способом.
9. Напишите формулу компактной записи функции принадлежности.
10. Сформулируйте определение лингвистической переменной.
11. Опишите набор переменных, с помощью которого описывается лингвистической переменная.
12. Дайте определения следующих понятий: X – универсальное множество объектов, $\widetilde{X}$ – базовое терм-множество, G – синтакси-

ческие правила вывода (порождения) новых термов, Р – семантические правила.

13. Сформулируйте определение нечеткой логики.
14. Какие необходимые характеристики нечеткой логики ввел Л. Заде?
15. Что лежит в основе операций нечеткой логики?
16. Какие операции используются для моделирования основных логических связок И, ИЛИ над нечеткими множествами в нечеткой логике?
17. Сформулируйте определения триангулярной нормы и триангулярной конормы (t-норма и s-конорма) и докажите их двойственность. Приведите примеры.
18. Запишите композиционное правило Л. Заде.
19. Сформулируйте содержательное определение операции импликации.
20. Сформулируйте определение системы нечеткого логического вывода.
21. Какие объекты входят в систему нечеткого логического вывода?
22. Какие условия должны соблюдаться при построении правил на основе системы нечеткого логического вывода?
23. Перечислите пять способов реализации нечеткого логического вывода.
24. Подробно опишите реализацию нечеткого логического вывода с помощью алгоритма Мамдани. Нарисуйте схему процесса нечеткого вывода этого алгоритма.

Нечеткие временные ряды

1. Сформулируйте определение уровня временного ряда.
2. Сформулируйте определение нечеткого временного ряда.
3. Сформулируйте определение носителя нечеткой метки $\widetilde{x}_i$.
4. Приведите примеры прикладных задач обработки нечетких ВР.
5. Дайте определение нечеткому разбиению.

6. Опишите основные этапы алгоритма моделирования нечетких ВР в соответствии с нечеткой моделью Сонга.
7. В чем преимущество использования моделей нечетких ВР?
8. Что понимается под интеллектуальным анализом данных или Data Mining?
9. Какие задачи решаются на основе Data Mining?
10. Приведите виды темпорально-логических концептов ВР.
11. В чем заключается метод аппроксимации временного ряда на основе F-преобразования?

Нечеткая регрессия

1. Какие существуют подходы к построению моделей нечеткой линейной регрессии?
2. Какие существуют критерии для определения нечетких коэффициентов модели?
3. Какие вы знаете варианты методов на основе классификации «вход – выход»?

Генетические алгоритмы

1. Поясните происхождение термина «генетические алгоритмы».
2. Опишите сферу применения генетических алгоритмов.
3. Дайте определение гену в контексте генетических алгоритмов.
4. Дайте определение хромосоме в контексте генетических алгоритмов.
5. Дайте определение популяции в контексте генетических алгоритмов.
6. Дайте определение степени приспособленности в контексте генетических алгоритмов.
7. Дайте определение кроссовера в контексте генетических алгоритмов.
8. Дайте определение мутации в контексте генетических алгоритмов.
9. Перечислите методы отбора хромосом для кроссовера.
10. Расскажите о типовой схеме генетического алгоритма.

Нечеткая кластеризация

1. Дайте определение классификации.
2. Дайте определение кластеризации.
3. Поясните разницу между классификацией и кластеризацией.
4. Определите область применение кластеризации.
5. Определите цель алгоритма FCM-кластеризации.
6. Определите перечень входных данных для алгоритма FCM-кластеризации.
7. Перечислите шаги алгоритма FCM-кластеризации.
8. Почему кластеризация, проводимая с помощью алгоритма FCM, является нечеткой?
9. Поясните назначение параметра «степень нечеткости» в алгоритме FCM.
10. Что является выходными данными алгоритма FCM-кластеризации?

Искусственные нейронные сети и глубинное обучение

1. Какие достоинства и недостатки есть у ИНС по сравнению с Регрессией и Решающими Деревьями?
2. Сеть какого типа лучше использовать для прогнозирования?
3. Сеть какого типа можно использовать в условиях постоянного изменения данных, когда точной выборки еще не существует?

Тест «Искусственные нейронные сети»

1. Выберите верные утверждения:
   a) ИНС проще подобрать под любую нелинейную задачу. Все, что нужно сделать, это увеличивать число слоев пропорционально числу признаков
   b) ИНС позволяют обрабатывать более высокоуровневые признаки за счет нелинейной функции активации и последовательным слоям

c) По сравнению с Регрессией ИНС практически не подвержены Переобучению при любом количестве нейронов
d) С точки зрения математического аппарата ИНС – это комбинация полиномиальной регрессии высокого порядка и формулы Байеса
e) ИНС может аппроксимировать любую нелинейную непрерывную функцию, но это еще не гарантирует 100% сходимости на произвольных данных
f) ИНС в отличие от регрессии может хорошо обрабатывать высокую степень мультиколлинеартности и противоречивости в данных

2. Сеть какого типа лучше использовать для прогнозирования временных рядов?

a) Сверточную
b) ART MAP
c) Импульсную
d) MLP
e) Рекуррентную
f) Когнитрон

3. Сеть какого типа лучше использовать для обработки трехмерных сцен?

a) MLP
b) Рекуррентную
c) ART MAP
d) Сверточную
e) Когнитрон
f) Импульсную

4. Сеть какого типа лучше использовать для решения задачи классификации клиентов по одиночному вектору клиентских характеристик (с учетом того, что этот вектор содержит большое количество категориальных признаков)?

a) Автокодировщик
b) MLP

c) Когнитрон
d) ART MAP
e) Сверточную
f) Рекуррентную
g) Импульсную

5. Сеть какого типа можно использовать в условиях постоянного изменения данных, когда точной выборки еще не существует и сеть приходится постоянно дообучивать на новых классах?

a) MLP
b) Сверточную
c) Когнитрон
d) Рекуррентную
e) ART MAP
f) Автокодировщик
g) Импульсную

ПРАКТИЧЕСКИЕ ЗАДАНИЯ

Работа с файлом данных «Титаника»

Приведенные ниже задания основаны на данных 'titanic.csv', где содержатся сведения о пассажирах «Титаника».

Задачи:

1. Какое количество мужчин и женщин плыло на корабле?
2. Какой части пассажиров удалось выжить? Посчитайте долю выживших пассажиров. Ответ приведите в процентах (число в интервале от 0 до 100, знак процента не нужен), округлив до двух знаков.
3. Какую долю пассажиры первого класса составляли среди всех пассажиров? Ответ приведите в процентах (число в интервале от 0 до 100, знак процента не нужен), округлив до двух знаков.
4. Какого возраста были пассажиры? Посчитайте среднее и медиану возраста пассажиров. В качестве ответа приведите два числа через пробел.
5. Коррелируют ли число братьев/сестер/супругов с числом родителей/детей? Посчитайте корреляцию Пирсона между признаками SibSp и Parch.
6. Какое самое популярное женское имя на корабле? Извлеките из полного имени пассажира (колонка Name) его личное имя (First Name). Это задание – типичный пример того, с чем сталкивается специалист по анализу данных. Данные очень разнородные и шумные, но из них требуется извлечь необходимую информацию. Попробуйте вручную разобрать несколько значений столбца Name и выработать правило для извлечения имен, а также разделения их на женские и мужские.

Работа по отбору признаков

Доработайте код отбора признаков, использовав все модели из списка:

1. Линейная регрессия (LinearRegression)
2. Гребневая регрессия (Ridge)
3. Лассо (Lasso)
4. Случайное Лассо (RandomizedLasso)
5. Рекурсивное сокращение признаков (Recursive Feature Elimination – RFE)
6. Сокращение признаков Случайными деревьями (Random Forest Regressor)
7. Линейная корреляция (f_regression)

Отобразите получившиеся значения\оценки каждого признака каждым методом\моделью и среднюю оценку. Проведите анализ получившихся результатов. Какие 4 признака оказались самыми важными по среднему значению? (Названия\индексы признаков и будут ответом на задание).Какие выводы можно сделать по результатам отдельных методов?

Многослойный персептрон

1. Доработайте код задачи классификации (листинг 58). Попробуйте настроить различные параметры сети, такие как количество нейронов в первом или втором слое, вид функции активации (activation), алгоритм градиентного спуска (solver), количество максимальных итераций при обучении (max_iter), порог точности при обучении (tol). Варьируя эти параметры, попробуйте найти наилучший вариант (возможные значения переменных можно найти в документации Scikit-learn).

2. Насколько лучше или хуже работает MLPClassifier по сравнению с персептроном? Без нормализации данных и с нормализацией данных? Предположите, почему именно так ведут себя модели.

3. Проведите серию экспериментов с различными значениями random_state. Как случайная инициализация весов влияет на Персептрон и многослойную сеть?

4. Дополнительно постройте график распределения исходных данных.

Реализация алгоритма обратного распространения ошибки

1. Модифицируйте сеть из листинга 59. Добавьте еще один слой и восстановите работоспособность алгоритма. Критерием его работоспособности может быть стремительно уменьшающаяся ошибка.

2. Дополнительно обучите старую и новую модели на следующих данных:

```
X = [[0, 0, 1], [0.3, 1, 0],
    [1, 0.3, 0],  [0.6, 0.2, 1],
    [0.6, 0.2, 1]]
```

Проанализируйте исходные данные. Какая зависимость скрыта в них? Объясните, что произошло с качеством классификации. Почему? С чем может быть связано подобное в реальных задачах и как на это можно повлиять (если это вообще возможно)?

Регуляризация и сеть прямого распространения

1. Вернитесь к задаче из пункта «Влияние регуляризации на многослойную сеть прямого распространения». Проанализируйте результат кода при количестве нейронов, равном 100, в скрытом слое MLP. Напомним, что регуляризация – это механизм снижения больших весов модели при повышении ее сложности. Этот механизм помогает избежать переобучения. Параметр регуляризации alpha регулирует размер штрафов, которые накладываются на веса. Какой коэффициент регуляризации оптимален для данной задачи?

2. Задача сравнения персептрона, многослойной сети и регрессий. В пункте «Влияние регуляризации на многослойную сеть прямого распространения» была описана общая методика по проведению

серии экспериментов и выводу результатов (понятно, что это можно сделать не только в виде графиков, но и выводом информации в консоль или записи более детальной информации в файл\таблицу для дальнейшей обработки). Соответственно каркас этой программы можно повторно использовать во многих других экспериментах не только по анализу влияния регуляризации.

По аналогии с заданием выше сгенерируйте 3 задачи классификации со следующими параметрами: make_moons (noise=0.3, random_state=rs), make_circles (noise=0.2, factor=0.5, random_state=rs), X, y = make_classification (n_samples=500, n_features=2, n_redundant=0, n_informative=2, random_state=rs, n_clusters_per_class=1) и сравните следующие модели:

- Линейную регрессию
- Полиномиальную регрессию (например, со степенью 3)
- Гребневую полиномиальную регрессию (например, со степенью 4, alpha = 1.0)
- Персептрон
- Многослойный персептрон с десятью нейронами в скрытом слое (alpha = 0.01)
- Многослойный персептрон со ста нейронами в скрытом слое (alpha = 0.01)

Создайте отдельный файл и замените только код моделей. Постройте графики и отобразите качество моделей. В результате у вас должно получиться полотно графиков, похожее на рисунок 71 (причем в данном случае представлена только первая задача классификации на Moondataset).

Проанализируйте полученные результаты, рассчитав их при параметре random_state=25 для следующих функций\объектов:

- Perceptron
- MLPClassifier
- Ridge
- make_classification

- rng = np.random.RandomState(25)
- make_moons
- make_circles
- train_test_split

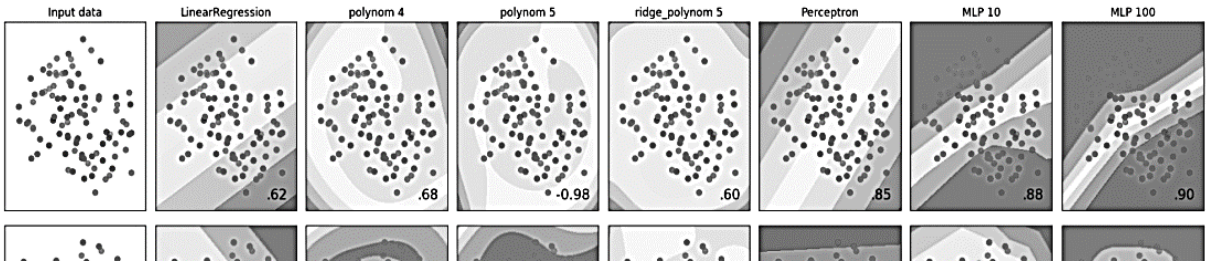


Рисунок 71. Пример работы программы по сравнению нескольких моделей на задачах moon, circle, linear

Сравнение эффективности моделей из библиотеки Keras

Вы научитесь:

- Проводить самостоятельный анализ результатов экспериментов;
- Проводить серию экспериментов;
- Интерпретировать результаты моделей;
- Подбирать соответствующие данным инструменты и параметры.

Необходимые import для выполнения работы представлены в листинге 95.

Листинг 95

```
import numpy as np
from keras.datasets import mnist
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten
from keras.layers import Convolution2D, MaxPooling2D
from keras.utils import np_utils
import matplotlib.pyplot as plt
from keras.models import model_from_json
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
```

Инструкции по выполнению представлены в листинге 96 (первые 3 шага), а также ниже. Прежде чем перейти к шагу 3, сравните качество полученных трех моделей на данных MNIST. Объясните полученные результаты. Есть ли превосходство сверточной сети перед обычной или перед полиномиальной регрессией?

Листинг 96

```
#Шаг 1. Загрузите данные MNIST
(X_train, y_train), (X_test, y_test) = mnist.load_data()
#Шаг 2. Создайте модель многослойного персептрона (на основе Keras)и сверточной
#сети (на основе Keras) и полиномиальной регрессии из Scikit-Learn.
#Для задания обычного линейного слоя в качестве первого слоя сети потребуется
#указать форму входного вектора (размер). Это можно сделать следующим образом:
Dense(128, input_dim=input_size)
#Шаг 3. Далее попробуйте эти три типа модели уже на других данных.
#Для этого загрузите и подготовьте данные с фотографиями людей с помощью
#библиотеки scikit-learn (изучите код самостоятельно):
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape
X = lfw_people.images
# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
print("n_classes: %d" % n_classes)
# split into a training and testing set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train = X_train.reshape(X_train.shape[0], 1, img_rows, img_cols)
X_test = X_test.reshape(X_test.shape[0], 1, img_rows, img_cols)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_train /= 255
X_test /= 255
```

Шаг 4. Отобразите данные, чтобы удостовериться в их корректной подготовке для дальнейшего обучения.

Шаг 5. Обучите модель многослойного персептрона, полиномиальной регрессии и сверточной сети. Попробуйте разное число нейронов и слоев в сетях (10, 20, 50, 100, 200, 500 нейронов) и (1-3 слоя). Большее число нейронов и слоев не нужно.

Замечание 1: Конечно же, не следует обучать модели сперва на MNIST, а потом эти же модели (конкретные экземпляры) на фотографиях. Это совсем разные задачи, и при малых количествах слоев и таких выборках вряд ли произойдет хорошее обобщение. Имеется в виду, что те же три типа моделей теперь надо сравнить на фотографиях.

Замечание 2: Если на CPU обучение идет слишком долго, то возьмите не всю обучающую выборку, а лишь половину или одну треть.

В заключение сравните качество обученных моделей на данных фотографий. Объясните полученные результаты и ответьте а вопросы:

- Почему получилось именно так?
- На что способны и не способны используемые модели?
- Чем отличаются данные из первой части задания от данных из второй части задания?

Работа с библиотекой OpenCV

Докажите гипотезу, сформулированную по завершению предыдущего задания. Что нужно сделать с данными MNIST, чтобы работающая ранее модель стала давать на них плохие результаты? Для этого вам может понадобиться библиотека OpenCV для Python. Которая достаточно легко устанавливается. Смотрите раздел «**Installing OpenCV from prebuilt binaries»** по следующей ссылке: [65].

Согласно Википедии [1]: «OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно

использоваться в академических и коммерческих целях – распространяется в условиях лицензии BSD.

Проект OpenCV возник в 1999 году в рамках исследования Intel по высокопроизводительным приложениям по обработке 3D сцен. Это крайне мощная библиотека и стандарт де-факто для обработки изображений. Сферы применения OpenCV:

Инструмент выделения 2D и 3D признаков;

Трехмерная навигация;

Системы распознавания лиц;

Системы распознавания жестов;

Человеко-компьютерные интерфейсы;

Мобильные роботы;

Распознавание эмоций;

Идентификация объектов;

Сегментация и распознавание;

Стереозрение;

Восстановление поверхности;

Отслеживание движения;

Дополненная реальность.

Кроме того, для поддержки некоторых вышеуказанных областей, OpenCV включает в себя библиотеки статистической обработки данных и машинного обучения:

Решающие деревья;

Алгоритм максимального правдоподобия;

Алгоритм k-ближайших соседей;

Наивный Байесовский классификатор;

Искусственные нейронные сети;

Случайные леса;

Метод опорных векторов.

В библиотеке OpenCV есть методы для многих видов деформаций изображений. Для базовой проверки способностей ИНС на двумерных изображениях вполне хватит следующих:

Повороты;

Линейные сдвиги;

Масштабирование».

Ниже (в листинге 97) приведен код, который позволяет выполнить преобразование исходного изображения images[i]. Каждый метод преобразования cv2 возвращает новое изображение. Для поворотных и линейных сдвигов используются специальные матрицы деформации. Код для их инициализации также приведен ниже. Недостающие переменные – это коэффициенты и константы. Определите их самостоятельно.

Листинг 97

```
import cv2
# setrotation
rot_Matrix = cv2.getRotationMatrix2D((cols / 2, rows/ 2), ang, 1)
cv2.warpAffine(images[i], rot_Matrix, (cols, rows))
# set scaling
cv2.resize(images[i], dsize=(rows, cols), fx=scale, fy=scale,
interpolation=cv2.INTER_CUBIC)
# set translation
trans_Matrix = np.float32([[1, 0, tx], [0, 1, ty]])
cv2.warpAffine(images[i], trans_Matrix, (cols, rows))
```

После преобразования части изображений проведите соответствующие эксперименты с разными моделями. Измерьте качество получившихся моделей. Вероятно, придется проделать много экспериментов с различными параметрами, чтобы добиться такой ситуации, когда результаты будут объяснять теорию.

Нечеткая логика

1. На языке Python разработайте скрипт, позволяющий задать нечеткое множество с треугольной функцией принадлежности и отобразить его название, параметры, а также степень принадлежности вводимого пользователем объекта.

2. На языке Python разработайте скрипт, позволяющий задать нечеткое множество с трапециевидной функцией принадлежности и отобразить его параметры, а также степень принадлежности вводимого пользователем объекта.

3. На языке Python разработайте скрипт, позволяющий выполнить операцию пересечения заданных пользователем нечетких множеств с треугольными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – пересечение данных нечетких множеств.

4. На языке Python разработайте скрипт, позволяющий выполнить операцию пересечения заданных пользователем нечетких множеств с трапециевидными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – пересечение данных нечетких множеств.

5. На языке Python разработайте скрипт, позволяющий выполнить операцию объединения заданных пользователем нечетких множеств с треугольными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – объединение данных нечетких множеств.

6. На языке Python разработайте скрипт, позволяющий выполнить операцию объединения заданных пользователем нечетких множеств с трапециевидными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – объединение данных нечетких множеств.

7. На языке Python разработайте скрипт, позволяющий выполнить операцию дополнения, заданного пользователем нечеткого множества с треугольной функцией принадлежности. Входными данными будут параметры функции принадлежности и четкие объекты множества. Выходными – дополнение нечеткого множества.

8. На языке Python разработайте скрипт, позволяющий выполнить операцию дополнения, заданного пользователем нечеткого множества с трапециевидной функцией принадлежности. Входными данными будут параметры функции принадлежности и четкие объекты множества. Выходными – дополнение нечеткого множества.

9. На языке Python разработайте скрипт, позволяющий выполнить операцию импликации заданных пользователем нечетких множеств с треугольными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – результат импликации данных нечетких множеств. Причем результат вывести через лингвистические переменные. Импликацию моделировать минимумом.

10. На языке Python разработайте скрипт, позволяющий выполнить операцию импликации заданных пользователем нечетких множеств с трапециевидными функциями принадлежности. Входными данными будут параметры функций принадлежности и четкие объекты для каждого из множеств. Выходными – результат импликации данных нечетких множеств. Импликацию моделировать минимумом.

Генетические алгоритмы

1. На языке Python разработайте скрипт, который с помощью генетического алгоритма решает следующую задачу. Дано N наименований продуктов, для каждого из которых известно m характеристик. Необходимо получить самый дешевый рацион из k наименований, удовлетворяющий заданным медицинским нормам для каждой из m характеристик.

2. На языке Python разработайте скрипт, который с помощью генетического алгоритма решает следующую задачу. Дано n пунктов производства продуктов и k городов, которые в них нуждаются. Каждый город может потребить x продуктов, а каждый пункт произвести y продуктов. Необходимо получить оптимальный маршрут, так, чтобы все города получили нужный им объем продуктов без сильного его превышения, а транспортные расходы были минимальными.

3. На языке Python разработайте скрипт, который с помощью генетического алгоритма решает следующую задачу. Дано N наименований продуктов, для каждого из которых известно m характеристик. Необходимо получить самый лучший по характеристикам рацион из k наименований, удовлетворяющий заданным ценовым рамкам. Лучшим считается рацион с минимальным отклонением от нормы.

4. На языке Python разработайте скрипт, который с помощью генетического алгоритма решает следующую задачу. Дано n пунктов производства продуктов и k городов, которые в них нуждаются. Каждый город может потребить x продуктов, а каждый пункт произвести y продуктов. Необходимо получить оптимальный маршрут, так, чтобы все города получили нужный им объем продуктов с минимальным его превышением, а транспортные расходы укладывались в определенные рамки.

5. На языке Python разработайте скрипт, который с помощью генетического алгоритма решает следующую задачу. Дано N полей и k культур для посева. Для каждого поля известна характеристика урожайности каждой из k культур, а для каждой культуры – его закупочная стоимость. Необходимо получить самый лучший урожай за наименьшую стоимость.

Нечеткая кластеризация объектов

1. На языке Python разработайте скрипт, кластеризующий загруженные данные о размере заработной платы n людей на определенные им кластеры, обозначенные заданными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

2. На языке Python разработайте скрипт, кластеризующий загруженные данные о возрасте n людей на определенные им кластеры, обозначенные заданными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

3. На языке Python разработайте скрипт, кластеризующий загруженные данные о стоимости n автомобилей на определенные им

кластеры, обозначенные заданными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

4. На языке Python разработайте скрипт, кластеризующий загруженные данные о росте *n* людей на определенные им кластеры, обозначенные заданными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

5. На языке Python разработайте скрипт, кластеризующий загруженные данные о размере затрат на производство *n* продуктов на определенные им кластеры, обозначенные определенными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

6. На языке Python разработайте скрипт, кластеризующий загруженные данные о длительности перелетов до *n* пунктов на определенные им кластеры, обозначенные заданными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

7. На языке Python разработайте скрипт, кластеризующий загруженные данные о размере затрат на связь среди *n* отделов на определенные им кластеры, обозначенные определенными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

8. На языке Python разработайте скрипт, кластеризующий загруженные данные о размере площадей *n* квартир на определенные им кластеры, обозначенные определенными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

9. На языке Python разработайте скрипт, кластеризующий загруженные данные о размере урожая *n* сельскохозяйственных культур на определенные им кластеры, обозначенные определенными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

10. На языке Python разработайте скрипт, кластеризующий загру-женные данные о весе *n* людей на определенные им кластеры,

обозначенные определенными в программе лингвистическими метками. Максимальное количество меток задать самостоятельно.

Анализ временных рядов

1. *Лингвистические шкалы.* Разработать скрипт на Python, позволяющий задавать с использованием функций принадлежности лингвистическую шкалу для решения задачи по варианту (таблица 14). Количество оценок в шкале и параметры функций принадлежности должны задаваться по умолчанию, если в заданном пользователем файле не указано иное. Осуществить графическое отображение функций принадлежности меток шкалы, выделив каждую отдельным цветом. Предусмотреть загрузку файла с данными и оценку их по построенной шкале.

Таблица 14. Варианты для задачи «Лингвистические шкалы»

Вариант	**Функция принадлежности**	**Назначение шкалы**
1.	Треугольная	Оценка эффективности продаж
2.	Трапециевидная	Оценка эффективности операций с валютой
3.	Треугольная	Оценка прибыли
4.	Треугольная	Оценка затрат на оплату труда
5.	Трапециевидная	Оценка рентабельности
6.	Треугольная	Оценка объемов продаж
7.	Трапециевидная	Оценка объемов закупок
8.	Треугольная	Оценка загруженности сервера и сети
9.	Треугольная	Оценка уровня зарплаты
10.	Трапециевидная	Оценка убытков
11.	Треугольная	Оценка эффективности продаж
12.	Трапециевидная	Оценка эффективности операций с валютой
13.	Треугольная	Оценка прибыли

Окончание таблицы 14

Вариант	Функция принадлежности	Назначение шкалы
14.	Треугольная	Оценка затрат на оплату труда
15.	Трапециевидная	Оценка рентабельности
16.	Треугольная	Оценка объемов продаж
17.	Трапециевидная	Оценка объемов закупок
18.	Треугольная	Оценка загруженности сервера и сети
19.	Треугольная	Оценка уровня зарплаты
20.	Трапециевидная	Оценка убытков
21.	Треугольная	Оценка эффективности продаж
22.	Трапециевидная	Оценка эффективности операций с валютой
23.	Треугольная	Оценка прибыли
24.	Треугольная	Оценка затрат на оплату труда
25.	Трапециевидная	Оценка рентабельности
26.	Треугольная	Оценка объемов продаж
27.	Трапециевидная	Оценка объемов закупок
28.	Треугольная	Оценка загруженности сервера и сети
29.	Треугольная	Оценка уровня зарплаты
30.	Трапециевидная	Оценка убытков

2. *Нечеткие временные ряды и ряды нечетких тенденций*. Реализовать скрипт на языке Python, позволяющий загружать четкий временной ряд величины из задачи по лингвистическим шкалам (задача «Лингвистические шкалы»). С помощью построенной в предыдущей работе шкалы (задача «Лингвистические шкалы») построить из четкого временного ряда временной ряд нечетких значений и нечетких тенденций. Осуществить графическое отображение нечетких меток шкалы и нечетких тенденций, выделив каждую отдельным цветом.

3. *Кластеризация и лингвистические шкалы.* Язык реализации Python. Построить шкалу из задачи по лингвистическим шкалам (работа «Лингвистические шкалы») с помощью алгоритма fcm-клас-

теризации. Количество кластеров задается по умолчанию, но может редактироваться пользователем посредствам загрузки из файла. Каждый кластер – метка на шкале с присвоенным ему лингвистическим значением либо по умолчанию, либо пользователем. Осуществить графическое отображение меток шкалы, выделив каждую отдельным цветом.

3. *Прогнозирование значения временного ряда с помощью нейронной сети.* Реализовать прогноз следующего значения четкого временного ряда из работы «Нечеткие временные ряды и ряды нечетких тенденций» с помощью нейронной сети, но НЕ рекуррентного типа. Графически отобразить реальный ряд и прогнозное значение, а также проиллюстрировать результат тестового прогноза.

4. *Прогнозирование значения временного ряда с помощью модифицированного метода Сонга.* Реализовать прогноз следующего значения временного ряда из работы «Нечеткие временные ряды и ряды нечетких тенденций» с помощью метода, описанного в пункте «Пример моделирования временного ряда в нечетком подходе». Графически отобразить реальный ряд и прогнозное значение, а также проиллюстрировать результат тестового прогноза.

5. *F-преобразование.* Для временного ряда из работы «Нечеткие временные ряды и ряды нечетких тенденций» выделить тренд методом F-преобразования.

Работа с рекуррентными сетями

1. Модифицируйте код из листинга 91: поэкспериментируйте с разной структурой сети, разным количеством точек, по которым строится прогноз, и с разным количеством LSTM-блоков. Оцените качество работы сети в разных экспериментах. Интерпретируйте результаты.

2. Решите задачу «Прогнозирование значения временного ряда с помощью нейронной сети» из блока заданий «Анализ временных рядов» с помощью рекуррентной сети. Сравните и интерпретируйте результаты.

3. Объясните работу кода из листинга 98.

Листинг 98

```
import numpy
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.utils import np_utils

numpy.random.seed(7)
alphabet = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
char_to_int = dict((c, i) for i, c in enumerate(alphabet))
int_to_char = dict((i, c) for i, c in enumerate(alphabet))
seq_length = 1
dataX = []
dataY = []
for i in range(0, len(alphabet) - seq_length, 1):
  seq_in = alphabet[i:i + seq_length]
  seq_out = alphabet[i + seq_length]
  dataX.append([char_to_int[char] for char in seq_in])
  dataY.append(char_to_int[seq_out])
  print seq_in, '->', seq_out
X = numpy.reshape(dataX, (len(dataX), seq_length, 1))
X = X / float(len(alphabet))
y = np_utils.to_categorical(dataY)

model = Sequential()
model.add(LSTM(32, input_shape=(X.shape[1], X.shape[2])))
model.add(Dense(y.shape[1], activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, nb_epoch=500, batch_size=1, verbose=2)
scores = model.evaluate(X, y, verbose=0)
print("Model Accuracy: %.2f%%" % (scores[1]*100))
for pattern in dataX:
  x = numpy.reshape(pattern, (1, len(pattern), 1))
 x = x / float(len(alphabet))
  prediction = model.predict(x, verbose=0)
  index = numpy.argmax(prediction)
  result = int_to_char[index]
  seq_in = [int_to_char[value] for value in pattern]
  print seq_in, "->", result
```

4. Проведите два эксперимента, модифицируя код из листинга 92 следующим образом:

1. Изменив значение seq_length, сделав его равным 3 и изменив форму входных данных следующим образом:

X = numpy.reshape(dataX, (len(dataX), 1, seq_length))

2. Изменив значение seq_length, сделав его равным 3 и изменив форму входных данных следующим образом:

X = numpy.reshape(dataX, (len(dataX), 1, seq_length))

Интерпретируйте полученные результаты.

Заключение

Мы рассмотрели модели и алгоритмы, используемые в сфере машинного обучения для анализа данных, а также способы их реализации на языке Python с использованием библиотек Keras, Theano и других.

В настоящее время машинное обучение и область анализа данных являются весьма перспективными направлениями в ИТ. Задачей данной книги было познакомить вас с азами этого направления, рассмотрев наиболее важные модели, алгоритмы и их реализацию. Однако, чтобы стать специалистом в сфере анализа данных, недостаточно просто знать модели и инструменты. Необходимо понимать границы их применений и преимущества одного перед другим.

Данное учебное пособие лишь приоткрывает дверцу в мир машинного обучения, освещая небольшую его часть. Если же после работы с книгой вы поняли, что анализ данных – крайне интересная вещь, то вам непременно нужно двигаться дальше. Причем не только в изучении моделей, методов, алгоритмов и инструментов их реализации (что вы можете сделать, изучив приведенные в библиографическом списке источники), но и в совершенствовании своего мышления. Ведь пока искусственный интеллект еще не изобретен, всю семантическую нагрузку задач машинного обучения берет на себя разработчик-человек. Поэтому он непременно должен обладать острым аналитическим умом.

Перефразируя Шерлока Холмса, напомним, что «развитый мозг – это то, что всегда будет в цене». В связи с этим, закончим пособие поже-ланием: оттачивайте свой разум, покоряйте новые вершины, не останавливайтесь на достигнутом!

ГЛОССАРИЙ

DataMining – интеллектуальный анализ данных.

TimeSeries DataMining – интеллектуальный анализ временных рядов.

Аномалия временного ряда – новый, не типичный паттерн временного ряда.

Апостериорная вероятность – назначенная событию вероятность при условии наличия знаний, поддерживающих его наступление и полученных опытным путем.

Априорная вероятность – назначенная событию вероятность, при условии отсутствия знаний, поддерживающих его наступление.

Временной ряд (ВР) – последовательность упорядоченных в равноотстоящие моменты времени пар (момент_времени, значение_характеристики).

Генетические алгоритмы – адаптивные методы поиска, использующиеся для решения задач функциональной оптимизации.

Градиент – вектор, указывающий направление наибольшего возрастания некоторой величины, значение которой меняется в скалярном поле.

Дефаззификация – получение оптимального четкого значения по агрегированному нечеткому понятию.

Дисперсия – мера разброса элементов выборки относительно ее математического ожидания.

Задача классификации – распределение некоторого множества объектов по заданному множеству групп (классов).

Задача кластеризации – разделение некоторого множества объектов на непересекающиеся группы (кластеры) таким образом, чтобы каждая группа состояла из схожих объектов, а объекты разных кластеров существенно отличались.

Задача регрессии – приближение неизвестной целевой зависимости на некотором множестве данных.

Задача таксономии – задача построения древообразной иерархической структуры, упорядочивающей исходные данные.

Закон распределения – функция, определяющая для выборки вероятность попадания в некоторый интервал или вероятность получения определенного значения.

Знание – совокупность утверждений о закономерностях и свойствах процессов и явлений, а также связывающих их правил логического вывода и правил использования их при принятии решений.

Индексирование объектов временного ряда – построение индексов для эффективного выполнения запросов к базам данных ВР.

Классификация объектов временного ряда – назначение ВР или их паттернам одного из заранее определенных классов.

Кластеризация объектов временного ряда – поиск группировок ВР или их паттернов.

Комбинированная модель прогнозирования – модель прогнозирования, состоящая из нескольких индивидуальных (частных) моделей, называемых базовым набором моделей.

Концепт временной продолжительности – присутствие определенного паттерна или признака ВР на определенном интервале времени.

Концепт нечеткости ВР – нечеткость выраженности темпоральных событий и отношений.

Концепт одновременности ВР – совпадение во времени темпоральных событий (паттернов различных ВР).

Концепт очередности ВР – порядок следования паттернов ВР во времени.

Кроссовер – операция в генетическом алгоритме, позволяющая хромосомам обмениваться между собой своими частями.

Лингвистическая переменная – это переменная, значениями которой являются слова или высказывания естественного или искусственного языка.

Математическое ожидание – среднее значение вероятностных элементов выборки.

Медиана – число, характеризующее выборку по среднему из ее значений.

Метод градиентного спуска – нахождение локального экстремума (минимума или максимума) функции путем движения вдоль градиента в направлении наискорейшего спуска, задаваемого антиградиентом.

Мутация – операция в генетическом алгоритме, произвольным образом изменяющая хромосому.

Недообучение – ситуация, когда алгоритм при обучении с учителем не дает удовлетворительно малой средней ошибки на обучающем множестве.

Нечеткая логика – логика, оперирующая нечеткими высказываниями и рассуждениями на базе частичной истинности.

Нечеткий временной ряд (НВР) – упорядоченная в равноотстоящие моменты времени последовательность наблюдений над некоторым процессом, состояния которого изменяются во времени, если значение состояния процесса в каждый момент времени может быть выражено с помощью нечеткой метки .

Нечеткое множество – совокупность объектов, принадлежащих ему с определенной степенью.

Нормализация данных – процесс масштабирования вектора каждого признака к такому виду, что вектор будет иметь единичную норму (при этом есть разные способы оценки\подсчета нормы).

Обобщающая способность – свойство модели отражать исходные данные в требуемые результаты (X –> Y) на всем множестве исходных данных (во всех сценариях, а не только на тренировочных примерах).

Обучение – способность алгоритма при решении задач некоторого класса выдавать на некотором опыте лучшие результаты в смысле заданной меры качества при предъявлении нового опыта.

Ошибка – численно выраженная разница между ответом модели и требуемым (реальным) значением.

Переобучение – свойство натренированного алгоритма на объектах тренировочной выборки давать существенно меньшую вероятность ошибки, чем на объектах тестовой.

Пространство признаков – это N-мерное пространство, где N – число измеряемых характеристик объектов, выделенное для конкретной задачи.

Регуляризация – добавление некоторой дополнительной информации к условию минимизации ошибки.

Резюмирование временного ряда – формирование краткого описания ВР, содержащего существенные черты с точки зрения решаемой задачи.

Сегментация временного ряда – разбиение временного ряда на значимые сегменты.

Система нечеткого логического вывода – модель, описывающая поведение систем на естественном (или близком к естественному) языке в виде приближенных рассуждений на основе композиционного правила вывода.

Среднеквадратическое отклонение – величина, характеризующая рассеивание значений выборки относительно ее математического ожидания.

Стандартизация данных – процесс приведения вектора каждого признака к такому виду, что его математическое ожидание станет нулевым, а дисперсия – единичной.

Степень принадлежности – значение, задаваемое функцией принадлежности и находящееся в интервале от 0 до 1.

Фаззификация – процесс установки соответствия между четким значением x и нечетким f через функцию принадлежности.

Функция принадлежности – функция, отображающая базовое значение x и нечеткое f в интервал от 0 до 1.

Хромосома (в генетическом алгоритме) – битовая строка, описывающая решение.

Частотный анализ временного ряда – поиск часто проявляющихся паттернов ВР.

Энтропия – мера неупорядоченности системы.

Предметный указатель

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. [Электронный ресурс]: Материалы свободной энциклопедии «Википедиа». URL: http://ru.wikipedia.org/wiki/, (режим доступа – свободный), (дата обращения: 28.03.2017).

2. [Электронный ресурс]: Материалы открытого курса по машинному обучению от компании ODS. URL: https://habrahabr.ru/company/ods/blog/325654/, (режим доступа – свободный), (дата обращения: 28.03.2017).

3. Воронина, В. В. Разработка приложений для анализа слабоструктурированных информационных ресурсов : учебное пособие / В. В. Воронина, В. С. Мошкин. – Ульяновск : УлГТУ, 2015. – 162 с.

4. [Электронный ресурс]: Материалы сайта machinelearning. URL: http://www.machinelearning.ru/wiki/index.php?title=Переобучение, (режим доступа – свободный), (дата обращения: 28.03.2017).

5. [Электронный ресурс]: Статья по регуляризации. URL: https://ru.wikipedia.org/wiki/Регуляризация_(математика), (режим доступа – свободный), (дата обращения: 28.03.2017).

6. [Электронный ресурс]: Материалы сайта MSDN. URL: https://msdn.microsoft.com/ru-ru/magazine/dn904675.aspx, (режим доступа – свободный), (дата обращения: 28.03.2017).

7. [Электронный ресурс]: Материалы сайта machinelearning. URL:http://www.machinelearning.ru/wiki/index.php?title=Нормальное_распределение, (режим доступа – свободный), (дата обращения: 28.03.2017).

8. Паклин, Н. Б. Глава 9, // Бизнес-аналитика: от данных к знаниям : учебное пособие / Н. Б. Паклин, В. И. Орешков. – 2-е изд.. – СПб. : Питер, 2013. – С. 444-459.

9. [Электронный ресурс]: Ю. Лаходюк, Энтропия и деревья принятия решений. URL: https://habrahabr.ru/post/171759/, (режим доступа – свободный), (дата обращения: 28.03.2017).

10. [Электронный ресурс]: Статья по деревьям решений. URL: http://ru.wikipedia.org/wiki/Дерево_принятия_решений, (режим доступа – свободный), (дата обращения: 28.03.2017).

11. Клячкин, В. Н. Статистические методы анализа данных / В. Н. Клячкин, Ю. Е. Кувайскова, В. А. Алексеева. – М. : Финансы и статистика, 2016. – 240 с.

12. Клячкин, В. Н. Статистические методы в управлении качеством: компьютерные технологии / В. Н. Клячкин. – М. : Финансы и статистика, ИНФРА-М, 2009. – 304 с.

13. Валеев, С.Г. Регрессионное моделирование при обработке наблюдений / С. Г. Валеев. – М. : Наука, 1991. – 272 с.

14. [Электронный ресурс]: Материалы свободной энциклопедии «Википедиа»: Статья под названием «Робастные методы». URL: http://ru.wikipedia.org/wiki/Робастность, (режим доступа – свободный), (дата обращения: 08.07.2017).

15. Zadeh, A. Lotfi. Fuzzy Sets / Lotfi A. Zadeh // Information and Control. – 1965.

16. Заде, Л. А. Основы нового подхода к анализу сложных систем и процессов принятия решений / Л. А. Заде // Математика сегодня. – М. : Знание, 1974. – С. 5-49.

17. Штовба, С. Д. Проектирование нечетких систем средствами MATLAB / С. Д. Штовба. – М. : Горячая линия – Телеком, 2007. – 288 с.

18. Ярушкина, Н. Г. Основы теории нечетких и гибридных систем : учебное пособие / Н. Г. Ярушкина. – М. : Финансы и статистика, 2004. – 320 с.

19. Ярушкина, Н. Г. Интеллектуальный анализ временных рядов : учебное пособие / Н. Г. Ярушкина, Т. В Афанасьева., И. Г. Перфильева. – М. : ИД «ФОРУМ» : ИНФРА-М, 2012. – 160 с. – (Высшее образование).

20. Song, Q. Fuzzy time series and its models / Q. Song, B. Chissom // Fuzzy Sets and Systems. – №54 (1993) – P. 269-277.

21. [Электронный ресурс] Дегтярев, К. Ю. Применение специализированных компьютерных программ и методов, основанных на нечетких временных рядах для краткосрочного прогнозирования USB/RUB котировок / К. Ю. Дегтярев. URL: http://www.exponenta.ru/educat/news/degtyarev/paper.pdf, (режим доступа – свободный), (дата обращения: 08.07.2017).

22. Batyrshin, I. Perception Based Time Series Data Mining for Decision Making / I. Batyrshin // IFSA’07 Fuzzy Logic, Soft Computing and Computational Intelligence.

23. [Электронный ресурс]: Материалы IRAFM-2015. URL: http://irafm.osu.cz/cif2015/main.php?c=Static&page=results, (режим доступа – свободный), (дата обращения: 08.07.2017).

24. Новак, В. Математические принципы нечеткой логики / В. Новак, И. Перфильева, И. Мочкорж; пер. с англ.; под ред. А. Н. Аверкина. – М. : ФИЗМАТЛИТ, 2006.

25. Афанасьева Т. В. Модель ACL-шкалы для генерации лингвистических оценок в принятии решений / Т. В. Афанасьева // Вопросы современной науки и практики. Университет им. В. И. Вернадского. Т. 2. Серия «Технические науки». – 2008. – №4 (14). – С. 91-97.

26. [Электронный ресурс]: Саймон Хайкин - Нейронные сети. Полный курс. URL: http://neuralnetworksanddeeplearning.com, (режим доступа – свободный), (дата обращения: 08.07.2017).

27. [Электронный ресурс]: Материалы сайта machinelearning. URL: http://www.machinelearning.ru/wiki/index.php?title=Модель_Мак-

Каллока_Питтса, (режим доступа – свободный), (дата обращения: 08.07.2017).

28. [Электронный ресурс]: Статья Логика мышления. Часть 3. Персептрон, сверточные сети. URL: https://geektimes.ru/post/214317/, (режим доступа – свободный), (дата обращения: 08.07.2017).

29. [Электронный ресурс]: Материалы сайта machinelearning. URL: http://www.machinelearning.ru/wiki/index.php?title=Самоорганизующаяся_карта_Кохонена, (режим доступа – свободный), (дата обращения: 08.07.2017).

30. [Электронный ресурс]: Материалы свободной энциклопедии «Википедиа»: Python. URL: https://ru.wikipedia.org/wiki/Python, (режим доступа – свободный), (дата обращения: 08.07.2017).

31. [Электронный ресурс]: Материалы по синтаксису Python. URL: https://habrahabr.ru/post/31180/, (режим доступа – свободный), (дата обращения: 08.07.2017).

32. [Электронный ресурс]: Официальный сайт разработчиков Anaconda. URL: https://www.continuum.io/Downloads, (режим доступа – свободный), (дата обращения: 08.07.2017).

33. [Электронный ресурс]: Официальный сайт PyCharm. URL: https://www.jetbrains.com/pycharm/download/#section=windows, (режим доступа – свободный), (дата обращения: 08.07.2017).

34. [Электронный ресурс]: Официальный сайт SciPy. URL: https://scipy.org/ (режим доступа – свободный), (дата обращения: 08.07.2017).

35. [Электронный ресурс]: Покоряем Python – уроки для начинающих. Функции lambda. URL: https://pythlife.blogspot.ru/2012/11/lambda.html (режим доступа – свободный), (дата обращения: 08.07.2017).

36. [Электронный ресурс]: Документация по DataFrame. URL: http://pandas.pydata.org/pandas-docs/stable/generated/pandas.Data

Frame.html (режим доступа – свободный), (дата обращения: 08.07.2017).

37. [Электронный ресурс]: Уроки по Pandas. URL: https://bitbucket.org/hrojas/learn-pandas (режим доступа – свободный), (дата обращения: 08.07.2017).

38. [Электронный ресурс]: Документация по Scikit-learn. URL: http://scikit-learn.org/stable/modules/preprocessing.html#preprocessing-normalization (режим доступа – свободный), (дата обращения: 08.07.2017).

39. [Электронный ресурс]: Сервис Kaggle. Массив данных о пассажирах Титаника URL: https://www.kaggle.com/prkukunoor/TitanicDataset (режим доступа – свободный), (дата обращения: 08.07.2017).

40. [Электронный ресурс]: Документация по Scikit-learn:Ridge. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge (режим доступа – свободный), (дата обращения: 08.07.2017).

41. [Электронный ресурс]: Ссылка для скачивания упоминаемых в книге файлов с данными. URL: https://drive.google.com/drive/folders/0B5yyS8oSQ0FDelpKRXg3c3lKVlU (режим доступа – свободный), (дата обращения: 20.07.2017).

42. [Электронный ресурс]: Документация по Scikit-learn:TFIDFVectorizer. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html#sklearn.feature_extraction.text.TfidfVectorizer (режим доступа – свободный), (дата обращения: 08.07.2017).

43. [Электронный ресурс]: Документация по Scikit-learn: DictVectorizer. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.DictVectorizer.html (режим доступа – свободный), (дата обращения: 08.07.2017).

44. [Электронный ресурс]: Документация по Scikit-learn: LogisticRegression. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (режим доступа – свободный), (дата обращения: 08.07.2017).

45. [Электронный ресурс]: Документация по Scikit-learn: LinearRegression. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html (режим доступа – свободный), (дата обращения: 08.07.2017).

46. [Электронный ресурс]: Документация по Scikit-learn:Lasso. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html (режим доступа – свободный), (дата обращения: 08.07.2017).

47. [Электронный ресурс]: Документация по Scikit-learn: MinMaxScaler. URL: http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html(режим доступа – свободный), (дата обращения: 08.07.2017).

48. [Электронный ресурс]: Функции map и zip и lambda. Python. URL: http://ninjaside.info/blog/ru/funkcii-map-i-zip-i-lambda-python/ (режим доступа – свободный), (дата обращения: 08.07.2017).

49. [Электронный ресурс]: Документация по Scikit-learn: RandomizedLasso. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.RandomizedLasso.html (режим доступа – свободный), (дата обращения: 08.07.2017).

50. [Электронный ресурс]: Документация по Scikit-learn:RFE. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html(режим доступа – свободный), (дата обращения: 08.07.2017).

51. [Электронный ресурс]: Документация по Scikit-learn: RandomForestRegressor. URL: http://scikit-learn.org/stable/modules/

generated/sklearn.ensemble.RandomForestRegressor.html (режим доступа – свобод-ный), (дата обращения: 08.07.2017).

52. [Электронный ресурс]: Документация по Scikit-learn: f_regression. URL: http://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html(режим доступа – свобод-ный), (дата обращения: 08.07.2017).

53. [Электронный ресурс]: Документация по Scikit-learn: Pipeline. URL: http://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html(режим доступа – свободный), (дата обращения: 08.07.2017).

54. [Электронный ресурс]: Документация по Scikit-learn: Cross-validation. URL: http://scikit-learn.org/stable/modules/cross_validation.html#cross-validation(режим доступа – свободный), (дата обращения: 08.07.2017).

55. [Электронный ресурс]: Документация по Scikit-learn: Cross_val_score. URL: http://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html, (режим доступа – свобод-ный), (дата обращения: 08.07.2017).

56. [Электронный ресурс]: Документация по Scikit-learn: Perceptron. URL: http://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Perceptron.html, (режим доступа – свободный), (дата обращения: 08.07.2017).

57. [Электронный ресурс]: Документация по Scikit-learn: MLPClassifier. URL: http://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html, (режим доступа – свобод-ный), (дата обращения: 08.07.2017).

58. [Электронный ресурс]: Официальный сайт NVIDIA. Проверка видеокарты. URL: https://developer.nvidia.com/cuda-gpus (режим доступа – свободный), (дата обращения: 08.07.2017).

59. [Электронный ресурс]: Официальный сайт NVIDIA. Скачивание CUDA. URL: https://developer.nvidia.com/cuda-downloads (режим доступа – свободный), (дата обращения: 08.07.2017).

60. [Электронный ресурс]: Официальный сайт NVIDIA. Скачивание CUDANN. URL: https://developer.nvidia.com/rdp/cudnn-download (режим доступа – свободный), (дата обращения: 08.07.2017).

61. [Электронный ресурс]: Документация по билиотеке Keras. URL: https://keras.io/layers/core/, (режим доступа – свободный), (дата обращения: 08.07.2017).

62. [Электронный ресурс]: Материалы сайта machinelearningmastery. URL: https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/, (режим доступа – свободный), (дата обращения: 08.07.2017).

63. [Электронный ресурс]: Статья по реализации генетического алгоритма на Python. URL: http://easydan.com/arts/2016/genetic-optimization/, (режим доступа – свободный), (дата обращения: 08.07.2017).

64. Воронина, В. В. Разработка веб-сервисов для анализа слабоструктурированных информационных ресурсов : учебное пособие / В. В. Воронина. – Ульяновск : УлГТУ, 2016. – 166 с.

65. [Электронный ресурс]: Документация «Installing OpenCV from prebuilt binaries». URL: http://docs.opencv.org/3.2.0/d5/de5/tutorial_py_setup_in_windows.html, (режим доступа – свободный), (дата обращения: 08.07.2017).

Учебное электронное издание

ВОРОНИНА Валерия Вадимовна
МИХЕЕВ Александр Вячеславович
ЯРУШКИНА Надежда Глебовна
СВЯТОВ Кирилл Валерьевич

ТЕОРИЯ И ПРАКТИКА МАШИННОГО ОБУЧЕНИЯ

Учебное пособие

Редактор Н. А. Евдокимова

ЛР № 020640 от 22.10.97.

Печатное издание
Подписано в печать 08.11.2017.
Усл. печ. л. 16.97. Тираж 100 экз. Заказ 932.

Ульяновский государственный технический университет,
432027, г. Ульяновск, ул. Сев. Венец, д. 32.

ИПК «Венец», УлГТУ, 432027, г. Ульяновск, ул. Сев. Венец, д. 32.
Тел.: (8422) 778-113