МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

для проведения лабораторных работ по дисциплине
«Интеллектуальный анализ данных»

Донецк
2021

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
ДОНЕЦКОЙ НАРОДНОЙ РЕСПУБЛИКИ
ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«ДОНЕЦКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

КАФЕДРА АВТОМАТИЗИРОВАННЫХ СИСТЕМ УПРАВЛЕНИЯ

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

для проведения лабораторных работ по дисциплине
«Интеллектуальный анализ данных»

для обучающихся по направлениям подготовки
09.03.01 «Информатика и вычислительная техника»,
09.03.02 «Информационные системы и технологии»,
02.03.02. «Фундаментальная информатика и информационные технологии»
заочной формы обучения

РАССМОТРЕНО
на заседании кафедры
«Автоматизированные системы управления»
Протокол № 4 от 16.11.2021 г.

УТВЕРЖДЕНО
на заседании учебно-издательского совета ДОННТУ
Протокол № 14 от 16.12.2021 г.

Донецк
2021

УДК 004.8(076)
ББК 32.813я73
М 54

Составитель:

Васяева Татьяна Александровна – кандидат технических наук, доцент, доцент кафедры автоматизированных системы управления.

М 54 **Методические рекомендации для проведения лабораторных работ по дисциплине «Интеллектуальный анализ данных»** [Электронный ресурс]: для обучающихся по направлениям подготовки 09.03.01 «Информатика и вычислительная техника», 09.03.02 «Информационные системы и технологии», 02.03.02. «Фундаментальная информатика и информационные технологии» заочной формы обучения/ ГОУВПО «ДОННТУ», каф. автоматизированных системы управления; сост. Т. А. Васяева. – Донецк: ДОННТУ, 2021. – Систем. требования: Acrobat Reader. – Загл. с титул. экрана.

Методические рекомендации разработаны с целью оказания помощи обучающимся в усвоении теоретического материала и получении практических навыков по дисциплине «Интеллектуальный анализ данных», которые содержат задания для проведения лабораторных работ по курсу.

УДК 004.8(076)
ББК 32.813я73

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина рассматривает вопросы получения данных из различных источников, контроля целостности и качества полученных данных, интеллектуального анализа данных, машинного обучения и глубокого обучения.

Цели дисциплины: ознакомление с теоретическими аспектами технологии Data Mining, Machine Learning and Deep Learning; ознакомление студентов с различными типами данных, методами их получения и оценкой качества; ознакомление с классическими задачами интеллектуального анализа данных; рассмотрение моделей для решения базовых задач интеллектуального анализа данных и изучение методов для их разработки; приобретение практических навыков по использованию инструментальных средств Data Mining, Machine Learning and Deep Learning.

В результате освоения дисциплины студент должен знать: место и значение анализа данных, основные этапы интеллектуального анализа данных; основные понятия, задачи, практическое применение, модели и методы интеллектуального анализа данных; принципы сбора данных из различных источников и их обработки; основы машинного и глубокого обучения; уметь: применять полученные знания при разработке, внедрении и эксплуатации автоматизированных и информационно-аналитических систем; получать данные из различных источников, выполнять их анализ и предварительную обработку; подбирать необходимые методы интеллектуального анализа в соответствии с задачами аналитической работы; выполнять программную реализацию методов интеллектуального анализа данных для решения типовых практических задач.

В методических указаниях приведены темы, необходимый теоретический и иллюстративный материал для выполнения лабораторных работ.

Лабораторная работа №1

Тема: применение библиотеки Scikit-Learn для решения задач классификации и регрессии.

Цель работы: изучение алгоритмов построения деревьев и правил решений; создание и исследование классификационных и регрессионных моделей с помощью деревьев и правил решений.

1.1. Постановка задачи классификации и регрессии

В задаче классификации и регрессии требуется определить значение зависимой переменной объекта на основании значений других переменных, характеризующих данный объект. Формально задачу классификации и регрессии описывают следующим образом. Имеется множество объектов:

$$I = \{i_1 , i_2 , \ldots, i_l,\ldots, i_n\}, \quad (1.1)$$

где i_j − исследуемый объект. Каждый объект характеризуется набором переменных:

$$I_j = \{x_1 , x_2 , \ldots, x_h,\ldots, x_m, y\}, \quad (1.2)$$

где x_h − независимые переменные, значения которых известны и на основании которых определяется значение зависимой переменной *y*. Набор независимых переменных обозначают в виде вектора:

$$X = \{x_1 , x_2 , \ldots, x_h,\ldots, x_m\}. \quad (1.3)$$

Каждая переменная x_h может принимать значения из некоторого множества:

$$C_h = \{c_{h1} , c_{h2}, \ldots\} \quad (1.4)$$

Если значениями переменной являются элементы конечного множества, то говорят, что она имеет категориальный тип. Если множество значений $C = \{c_{h1} , c_{h2}, \ldots, c_r, \ldots , c_k\}$ переменной *y* − конечное, то задача называется задачей классификации. Если переменная *y* принимает значение на множестве действительных чисел *R,* то задача называется задачей регрессии.

1.2. Структура дерева решений

Деревья решений – иерархические древовидные структуры, состоящие из решающих правил вида «если…то…» и позволяющие выполнить классификацию объектов. В дереве каждому объекту соответствует единственный узел, дающий решение.

В состав деревьев решений входят два вида объектов – узлы (node) и листья (leaf). В узлах содержатся правила, с помощью которых производится проверка атрибутов, и множество объектов в данном узле разбивается на подмножества. Листья – это конечные узлы дерева, в которых содержатся подмножества, ассоциированные с классами. Основным отличием листа от узла является то, что в листе не производится проверка, разбивающая ассоциированное с ним подмножество и, соответственно, нет ветвления.

Обычно входной узел называют корневым узлом (root node). Следовательно, дерево растет сверху вниз. Узлы и листья, подчиненные узлу более высокого иерархического уровня, называются потомками или дочерними узлами, а этот узел по отношению к ним – предком, или родительским узлом. Обобщенная структура дерева показана на рис. 1.1.

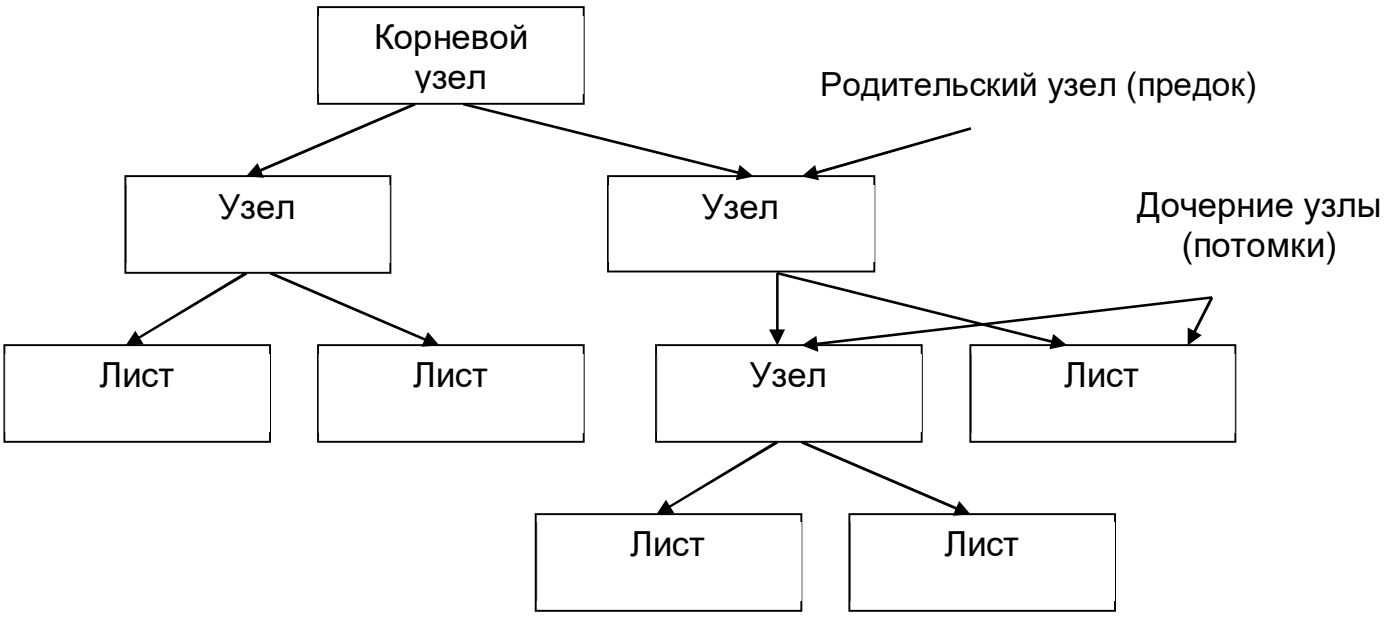


Рисунок 1.1 Узлы и листья в дереве решений

1.3. Построения деревьев решений

Дерево решений строится на основе обучающего множества, которое влияет на эффективность его построения. Анализируемые данные должны быть представлены в виде структурированного набора. Одно из полей обязательно должно содержать независимую переменную – метку класса. Обучающее множество должно содержать достаточно большое количество различных примеров.

Процесс роста дерева решений начинается с разбиения корневого узла на два потомка или более, каждый из которых рекурсивно подвергается дальнейшему разбиению. В процессе построения дерева для каждого узла необходимо выбирать атрибут, по которому будет производиться разбиение. Его принято называть атрибутом ветвления, или атрибутом разбиения (splitting attributed) и от того, насколько удачно выбираются атрибуты ветвления, зависит классифицирующая сила построенной модели.

При выборе атрибута ветвления: очередной выбранный атрибут должен обеспечивать наилучшее разбиение в узле. Наилучшим разбиением считается то, которое позволяет классифицировать наибольшее число примеров и создавать максимально чистые подмножества. Когда больше не удается обнаружить разбиения, значимо повышающие чистоту дочерних узлов, или когда число примеров в узле достигает некоторого заданного минимума, процесс разбиения для данной ветви заканчивается. Узел объявляется листом.

Очевидно, что процесс построения дерева решений не является однозначно определенным. Для различных атрибутов и даже для различного порядка их применения могут быть сгенерированы различные деревья решений. Большинство алгоритмов построения деревьев решений являются «жадными алгоритмами».

«Жадными» называются алгоритмы, которые на каждом шаге делают локально оптимальный выбор, допуская, что итоговое решение также окажется оптимальным. При построении деревьев решений «жадность» алгоритма

заключается в следующем: для каждого узла ищется оптимальный атрибут разбиения и предполагается, дерево в целом также будет оптимальным.

1.4. Критерии выбора наилучших атрибутов ветвления

Существует множество различных подходов к оценке потенциальных разбиений. При этом методы, разработанные в рамках теории машинного обучения, в основном сосредотачиваются на повышении чистоты результирующих подмножеств, в то время как статистические методы фокусируются на статистической значимости различий между распределением значений выходной переменной в узлах. Альтернативные методы разбиения часто приводят к построению совершенно разных деревьев, которые, впрочем, функционируют примерно одинаково.

Для выбора атрибута ветвления широко используются такие методы:

- индекс Джини, или метод разнообразия выборки (Gini-index);
- энтропия, или прирост информации (information gain);
- отношение прироста информации (gain-ratio);
- тест хи-квадрат (chi-square test).

Индекс Джини

Один из популярных критериев разбиения получил название индекса Джини в честь итальянского статистика и экономиста. Эта мера основана на исследовании разнообразия совокупности. Она определяет вероятность того, что два объекта, случайным образом выбранные из одной совокупности, относятся к одному классу. Очевидно, что для абсолютно чистой выборки данная вероятность равна 1.

Мера Джини для узла представляет собой простую сумму квадратов долей классов в узле. В случае, представленном на рис. 1.2, родительский узел содержит одинаковое количество светлых и темных кругов.

Для узла, в котором содержится равное число объектов, относящихся к двум классам, можно записать: $0{,}5^2 + 0{,}5^2 = 0{,}5$. Такой результат вполне ожидаем,

поскольку вероятность того, что случайно дважды будет выбран пример, относящийся к одному классу, равна 1/2. Индекс Джини для любого из двух результирующих узлов будет $0,1^2 + 0,9^2 = 0,82$. Идеально чистый узел имеет значение индекса Джини равное 1. Узел, в котором содержится равное число объектов двух классов будет иметь значение индекса 0,5. Таким образом, предпочтение следует отдать тому атрибуту, который обеспечит максимальное значение индекса Джини.

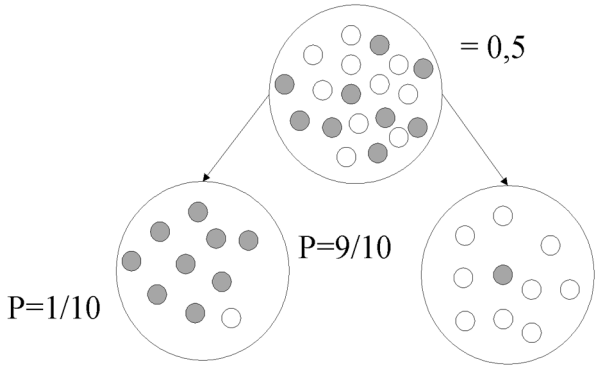


Рисунок 1.2. Пример разбиения

Уменьшение энтропии, или прирост информации

Из теории информации известно, что чем больше состояний может принимать некоторая система, тем сложнее ее описать и тем больше информации для этого потребуется. Аналогичный подход можно применить к описанию подмножества в некотором узле дерева решений. Если лист совершенно чистый, то все попавшие в него примеры относятся к одному классу и его описание будет очень простым. Но если лист содержит смесь объектов различных классов, то его описание усложнится. Существует мера количества информации – энтропия. Она отражает степень неупорядоченности системы.

В контексте нашего рассмотрения энтропия – это мера разнообразия классов в узле. Проще говоря, будем считать, что это мера, определяющая количество вопросов «да/нет», на которые нужно ответить, чтобы определить состояние системы. Если существует 16 возможных состояний, это даст $\log_2(16)$, или 4 бита, необходимых для описания всех состояний.

Целью разбиения узла в дереве решений является получение дочерних узлов с более однородным классовым составом. В результате разбиения должны образовываться узлы с меньшим разнообразием состояний выходной переменной. Следовательно, энтропия падает, а количество внутренней информации в узле растет. Уменьшение энтропии эквивалентно приросту информации.

Известно, что значение энтропии для некоторого множества максимально тогда, когда все классы в этом множестве равновероятны, то есть с точки зрения классификации имеет место полная неопределенность. Когда вероятность появления одного из классов увеличивается, энтропия множества уменьшается. При разбиении целесообразно выбрать атрибут, который позволит разбить множество так, чтобы в результирующих подмножествах определенный класс стал доминирующим, то есть алгоритм должен минимизировать энтропию.

Формально энтропия определенного узла T дерева решений определяется формулой:

$$Info(T) = \sum_{j=1}^{k} p_j \log_2 p_j \quad (1.5)$$

,

и представляет собой сумму всех вероятностей появления примеров, относящихся к j-му классу, умноженную на логарифм этой вероятности. Поскольку вероятность меньше или равна 1, значение логарифма всегда будет отрицательным. На практике эта сумма обычно умножается на – 1 для получения положительного числа.

Энтропия всего разбиения – это сумма энтропии всех узлов, умноженных на долю записей каждого узла в числе записей исходного множества.

Пусть в некотором узле дерева решений содержится множество T, которое состоит из N примеров. При этом неважно, является ли данный узел корневым или потомком. Тогда по формуле (1.5) для него может быть рассчитана энтропия $Info(T)$. В результате разбиения S для данного узла были созданы k потомков T_1, $T_2 \ldots T_k$, каждый из которых содержит число записей N_1, N_2 и N_k. Для потомков

рассчитывается энтропия по формуле (1.5): $Info(T_1)$, $Info(T_2)$… $Info(T_k)$. Тогда общая энтропия разбиения S:

$$Info(S) = \frac{N_1}{N} \cdot Info(T_1) + \frac{N_2}{N} \cdot Info(T_2) + ... + \frac{N_k}{N} \cdot Info(T_k) = \sum_{i=1}^{k} \frac{N_i}{N} \cdot Info(T_i) \quad (1.6)$$

Для примера, показанного на рис. 1.2, формула энтропии отдельного узла будет:

$$Info(T) = (-1) \cdot (P(темные) \cdot \log_2 P(темные) + P(светлые) \cdot \log_2 P(светлые)) \quad (1.7)$$

Здесь Р(темные) – вероятность того, что случайно выбранная из подмножества запись имеет значение целевой переменной темные. В нашем примере вероятность Р(темные) и Р(светлые) равны 0,5. Подставим значение 0,5 в предыдущую формулу и получим:

$$Info(T) = -1\ (0{,}5 \cdot \log_2 0{,}5 + 0{,}5 \cdot \log_2 0{,}5) = -1 \cdot \log_2 (0{,}5) = 1. \quad (1.8)$$

Что представляет энтропия узлов, полученных в результате разбиения. Один из них содержит 1 темный и 9 светлых объектов, в то время как другой – 1 светлый и 9 темных. Очевидно, что оба узла имеют одинаковую энтропию:

$$Info(T_1) = Info(T_2) = -1 \cdot (0{,}1 \cdot \log_2 0{,}1 + 0{,}9 \cdot \log_2 0{,}9) = 0{,}33 + 0{,}14 = 0{,}47. \quad (1.9)$$

Для вычисления общей энтропии разбиения воспользуемся формулой (1.6):

$$Info(S) = \frac{N_1}{N} \cdot Info(T_1) + \frac{N_2}{N} \cdot Info(T_2) = \frac{10}{20} \cdot 0{,}47 + \frac{10}{20} \cdot 0{,}47 = 0{,}47 \quad (1.10)$$

В результате разбиения уменьшение полной энтропии, или прирост информации, составит $Gain(S) = 1 - 0{,}47 = 0{,}53$. Это показывает, что данное разбиение является эффективным.

Отношение прироста информации

Поэтому деревья решений, которые строятся с использованием критерия уменьшения энтропии, предрасположены к сильной ветвистости. Сложные деревья с большим числом ветвей имеют узлы с малым количеством примеров, трудно интерпретируются, ведут к снижению устойчивости модели в целом. Для решения этой проблемы в рассмотрение включается дополнительный

показатель, который представляет собой оценку потенциальной информации, созданной при разбиении множества *T* на *n* подмножеств *Ti*:

$$Split - Info(T) = -\sum_{i=1}^{n}((|T_i|/|T|)) \cdot \log_2(|T_i|/|T|)) \quad (1.11)$$

С помощью этого показателя можно модифицировать критерий прироста информации, перейдя к отношению:

$$Gain - ratio = Gain(T)/(Split - Info(T)) \quad (1.12)$$

Такой критерий называется отношение прироста информации, он позволяет оценить долю информации, полученной при разбиении, которая является полезной, то есть способствует улучшению классификации.

Тест хи-квадрат

Тест хи-квадрат – это тест статистической значимости различия между распределениями значений двух выборок, разработанный К. Пирсоном. Тест представляет собой оценку вероятности того, что статистические законы распределения для двух неупорядоченных совокупностей наблюдений значительно различаются. При использовании чистоты в качестве меры эффективности разбиения, большое значение теста хи-квадрат говорит о том, что различие значимо.

1.5. Сложность и оптимизация модели

Для деревьев решений сложность модели пропорционально количеству узлов разбиений и листьев. Очень важно найти баланс между точностью и сложностью дерева. С этой целью разработан комплекс подходов и методов под общим названием упрощение деревьев решений, в англоязычной литературе известный как pruning. Обычно процесс построения дерева решений делится на две фазы: фазу роста и фазу упрощения.

Существуют два основных подхода к выбору оптимальной сложности дерева решений.

1. Принудительная остановка алгоритма с помощью условия, при выполнении которого рост дерева автоматически остановится. Это метод называется ранней остановкой (prepruning) (рис. 1.3). Здесь могут использоваться такие параметры, ограничивающие рост дерева, как минимальное количество примеров в узле, глубина дерева, статистическая зависимость разбиений и т.д.

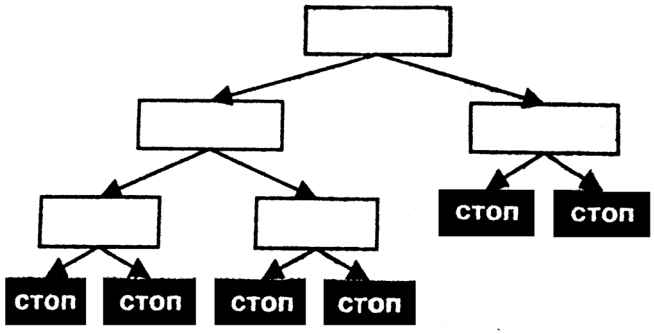


Рисунок 2.3. Ранняя остановка построения дерева решений

2. Сначала строится полное дерево, затем производится его упрощение путем отсечения ветвей (postrunning) (рис. 1.4). В данном случае создается последовательность поддеревьев в порядке увеличения их сложности. После этого при помощи определенного критерия выбирается лучшее поддерево. Обычно в качестве такого критерия используется эффективность работы дерева на валидационном множестве.

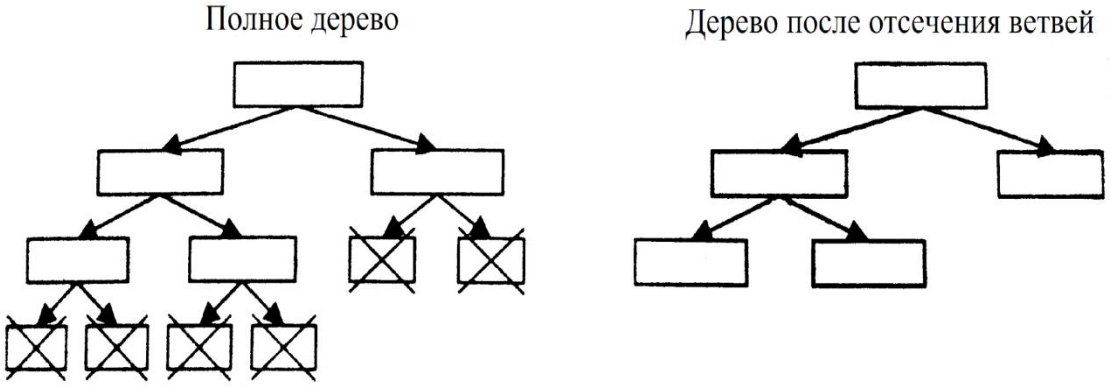


Рисунок 2.4. Отсечение ветвей

Упрощение деревьев решений преследует такие цели:

- уменьшить сложность дерева и извлеченных из него правил, повысить интерпретируемость классификационной модели;

- повысить устойчивость и обобщающую способность модели;
- сократить вычислительные затраты, связанные с работой модели.

Предварительная остановка менее затратная в вычислительном плане, но возникает риск потери хороших разбиений, которые могут следовать за плохими. Поэтому отсечение ветвей, как правило, позволяет получать лучшие результаты и пользуется большей популярностью.

2.6. Алгоритмы построения деревьев решений

Разработано достаточно много алгоритмов построения деревьев решений. Под алгоритмом построения дерева решений понимают метод, в соответствии с которым осуществляется выбор атрибута ветвления на каждом шаге.

Одними из наиболее популярных алгоритмов построения деревьев решений являются алгоритмы ID3 и его модификации C4.5, C5.0.

Для выбора атрибута разбиения ID3 использует критерий, называемый приростом информации (information gain), или уменьшением энтропии (entropy reduction), рассмотренный ранее. Основным недостатком алгоритма ID3 является тенденция к переобучению. Критерий прироста информации всегда будет приводить к выбору атрибутов с наибольшим количеством уникальных значений. Данная проблема решается использованием критерия, называемым отношение прироста информации, который рассмотрен выше. Использование данного критерия обычно приводит к выбору более удачного атрибута разбиения. Данная модификация получила название алгоритм C4.5.

CART (Classification and Regression Tree) − популярный алгоритм построения деревьев решений, предложенный в 1984 г. Деревья решений, построенные с помощью CART, являются бинарными, то есть содержат только два потомка в каждом узле.

В алгоритме CART идея нечистоты узла формализована с использованием индекса *Gini*. Если набор данных T содержит данные n классов, тогда индекс *Gini* определяется как:

$$Gini(T) = 1 - \sum_{j=1}^{n} p_j \quad (1.13)$$

где p_j – вероятность (относительная частота) класса j в T.

Если набор T разбивается на две части T_1 и T_2 с числом примеров в каждом N_1 и N_2 соответственно, тогда показатель качества разбиения будет равен:

$$Gini(T) = \frac{N_1}{N} Gini(T_2) + \frac{N_2}{N} Gini(T_2) \quad (1.14)$$

Наилучшим считается то разбиение, для которого *Gini(T)* минимально. Обозначим N – число примеров в узле – предке, L, R – число примеров соответственно в левом и правом потомке, l_i и r_i – число экземпляров i-го класса в левом/правом потомке. Тогда качество разбиения оценивается по следующей формуле:

$$Gini_{split} = \frac{L}{N}\left(1 - \sum_{i=1}^{n}\left(\frac{l_i}{L}\right)^2\right) + \frac{R}{N}\left(1 - \sum_{i=1}^{n}\left(\frac{r_i}{R}\right)^2\right) \to \min \quad (1.15)$$

Чтобы уменьшить объем вычислений формулу можно преобразовать:

$$Gini_{split} = \frac{1}{N}\left(L \bullet \left(1 - \frac{1}{L^2}\sum_{i=1}^{n} l_i^2\right)\right) + \frac{1}{N}\left(R \bullet \left(1 - \frac{1}{R^2}\sum_{i=1}^{n} r_i^2\right)\right) \to \min \quad (1.16)$$

Так как умножение на константу не играет роли при минимизации:

$$Gini_{split} = L - \frac{1}{L}\sum_{i=1}^{n} l_i^2 + R - \frac{1}{R}\sum_{i=1}^{n} r_i^2 \to \min \quad (1.17)$$

$$Gini_{split} = N - \left(\frac{1}{L}\sum_{i=1}^{n} l_i^2 + \frac{1}{R}\sum_{i=1}^{n} r_i^2\right) \to \min \quad (1.18)$$

$$\widetilde{G}_{split} = \frac{1}{L}\sum_{i=1}^{n} l_i^2 + R - \frac{1}{R}\sum_{i=1}^{n} r_i^2 \to \max \quad (1.19)$$

Иногда в алгоритме CART используются другие критерии разбиения.

На основе критерия хи-квадрата построен популярный алгоритм CHAID (Chisquare Automatic Interaction Detector). Принцип его работы основан на обнаружении статистических связей между переменными.

1.7. Понятие случайного леса

Случайный лес широко используется в машинном обучении, основан на построении большого числа (ансамбля) деревьев решений, каждое из которых строится по выборке, получаемой из исходной обучающей выборке с помощью бутстрепа, а на этапе расщепления вершин используется фиксированное число случайно отбираемых признаков. В качестве гиперпараметров можно использовать количество деревьев, а также гиперпараметры алгоритма построения дерева.

Бутстреп составляет класс методов генерации повторной выборки. Суть метода заключается в формировании множества выборок, на основе случайного выбора с повторениями.

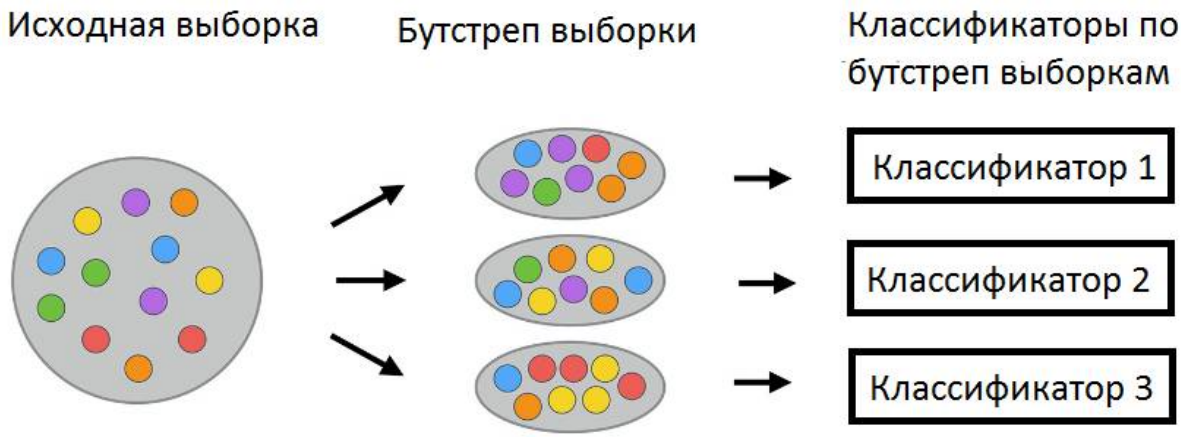


Рисунок 1.5. Бутстреп

Баггинг (bagging – bootstrap aggregation): обучение базовых правил происходит на различных случайных подвыборках данных или/и на различных случайных частях признакового описания; при этом базовые правила строятся независимо друг от друга.

Бустинг (boosting): каждое следующее базовое правило строится с использованием информации об ошибках предыдущих правил.

Алгоритм состоит из следующих шагов:

1. Для $i = 1, 2, \ldots, B$ (здесь B – количество деревьев в ансамбле) выполнить:

- сформировать бутстреп выборку S размера l по исходной обучающей выборке;
- по бутстреп выборке S построить дерево решений T_i .

2. В результате выполнения шага 1 получаем ансамбль деревьев решений $\{T_i\}^B_{i=1}$;

3. Предсказание новых наблюдений осуществляется усреднением для регрессии и голосованием для классификации.

1.8. Решающие правила

Альтернативный подход заключается в построении правил решений для каждого класса по отдельности.

Алгоритм Naive Bayes, использует формулу Байеса для расчета вероятности. Название naive (наивный) происходит от наивного предположения, что все рассматриваемые переменные независимы друг от друга. В действительности это не всегда так, но на практике все же данный алгоритм находит применение.

Вероятность того, что некоторый объект i_1 относится к классу c_r (т.е. $y = c_r$), обозначим как $P(y = c_r)$. Событие, соответствующее равенству независимых переменных определенным значениям, обозначим как E, а вероятность его наступления P(E). Идея алгоритма заключается в расчете условной вероятности принадлежности объекта к определенному классу c_r при равенстве его независимых переменных определенным значениям. Из теории вероятности известно, что ее можно вычислить по формуле:

$$P(y = c_r \mid E) = P(E \mid y = c_r) \cdot P(y = c_r) / P(E). \qquad (1.20)$$

Другими словами, формируются правила, в условных частях которых сравниваются все независимые переменные с соответствующими возможными значениями. В заключительной части присутствуют все возможные значения зависимой переменной:

$$\text{если } x_1 = c_h^1 \text{ и } x_2 = c_h^2 \text{ и } \ldots \; x_m = c_h^m \text{ тогда } y = c_r . \qquad (1.21)$$

Для каждого из этих правил по формуле Байеса определяется его вероятность. Предполагая, что независимые переменные принимают значения независимо друг от друга, выразим вероятность P(E|y= c_r) через произведение вероятностей для каждой независимой переменной:

$$P(E \mid y = c_r) = P(x_1 = c^1{}_p \mid y = c_r) \times P(x_2 = c^2{}_d \mid y = c_r) \times \ldots \times P(x_m = c^m{}_b \mid y = c_r). \quad (1.22)$$

Тогда вероятность для всего правила можно определить по формуле:

$$P(y = c_r \mid E) = P(x_1 = c^1{}_p \mid y = c_r) \times P(x_2 = c^2{}_d \mid y = c_r) \times \ldots \times P(x_m = c^m{}_b \mid y = c_r) \times P(y = c_r) / P(E). \quad (1.23)$$

Вероятность принадлежности объекта к классу cr при условии равенства его переменной x_h некоторому значению $c^h{}_d$ определяется:

$$P(x_h = c^h{}_d \mid y = c_r) = P(x_h = c^h{}_d \mid y = c_r)/P(y = c_r), \quad (1.24)$$

то есть равно отношению количества объектов в обучающей выборке, у которых x_h=$c_d{}^h$ и y = c_r к количеству объектов, относящихся к классу c_r.

1.9. Меры эффективности деревьев и правил решений

Классификационную способность дерева (правила) оценивают, рассчитывая точность и ошибку классификации. Точность классификации рассчитывается как отношение объектов, правильно классифицированных в процессе обучения, к общему количеству объектов набора данных, которые принимали участие в обучении. Ошибка рассчитывается как отношение объектов, неправильно классифицированных в процессе обучения, к общему количеству объектов набора данных, которые принимали участие в обучении.

1.10. Обзор методов классификации и регрессии в машинном обучении с помощью Python и Scikit-Learn

Для машинного обучения на Python написано очень много библиотек, одна из самых популярных – Scikit-Learn. Она упрощает процесс создания классификаторов, реализуя их с помощью понятной, хорошо документированной библиотеки.

Типы классификаторов в Scikit-Learn:

- метод опорных векторов (Support Vector Machines);
- классификатор дерева решений (Decision Tree Classifier)
- случайный лес (Random Forests);
- наивный байесовский метод (Naive Bayes);
- логистическая регрессия (Logistic Regression).

Это не все классификаторы, которые есть в Scikit-Learn. Про эти и другие можно прочитать в документации на сайте https://scikit-learn.org/stable/user_guide.html.

Классификаторы реализованы в виде классов. Для примера рассмотрим классификатор Classification and Regression Trees, который часто используется для задач как регрессии так и классификации. В таблице 1.1 представлены параметры этого класса. В таблице 1.2 представлены атрибуты этого класса. А в таблице 1.3 методы этого класса.

```
class sklearn.tree.DecisionTreeClassifier(
criterion='gini',
splitter='best',
max_depth=None,
min_samples_split=2,
min_samples_leaf=1,
min_weight_fraction_leaf=0.0,
max_features=None,
random_state=None,
max_leaf_nodes=None,
min_impurity_decrease=0.0,
min_impurity_split=None,
class_weight=None,
presort='deprecated',
ccp_alpha=0.0).
```

Таблица 1.1. Параметры классификатора Classification and Regression Trees

Параметр	Описание	Возможные значения	Значение по умолчанию
criterion	Критерии выбора наилучших атрибутов ветвления. Поддерживаемые критерии: индекс Джини, или метод разнообразия выборки и энтропия, или прирост информации.	gini, entropy	default=”gini”
splitter	Стратегия, используемая для выбора разделения на каждом узле. Поддерживаемые стратегии: «best» для выбора лучшего разбиения и «random» для выбора лучшего случайного разбиения.	best, random	default=”best”
max_depth	Максимальная глубина дерева. Если «None», то происходит разбиение, пока все листья не станут чистыми или пока все листья не будут содержать менее чем min_samples_split объектов.	int	default=None
min_samples_split	Минимальное количество объектов узле, необходимое для его разбиения. Если int, тогда рассматривается min_samples_split как минимальное число. Если float, то min_samples_split - это дробь, значение которой (min_samples_split * n_samples) - минимальное количество объектов для каждого разбиения.	int or float,	default=2
min_samples_leaf	Минимальное количество объектов, которое должно быть в листе. Узел может разбиваться на любой глубине только в том случае, если количество объектов в этом узле не менее чем min_samples_leaf	int or float	default=1

Параметр	Описание	Возможные значения	Значение по умолчанию
	обучающих объектов в каждой из левой и правой ветвей. Это может иметь эффект сглаживания модели, особенно в регрессии. Если int, тогда рассматривается min_samples_leaf как минимальное число. Если float, то min_samples_leaf - это дробь, значение которой (min_samples_leaf * n_samples) - минимальное количество объектов для каждого узла.		
min_weight_fraction_leaf	Минимальная взвешенная доля общей суммы весов (всех входных объектов), необходимая для конечного узла. Если объекты имеют одинаковый вес, когда sample_weight не указан.	float	default=0.0
max_features	Количество входных атрибутов, которые следует учитывать при поиске лучшего разделения: Если int, тогда рассматривается max_features при каждом разделении. Если float, тогда max_features - это дробь, int (max_features * n_features) рассматриваются при каждом разбиении. Если «авто», то max_features = sqrt (n_features). Если «sqrt», то max_features = sqrt (n_features). Если «log2», то max_features = log2 (n_features). Если нет, то max_features = n_features. Примечание. Поиск разбиения не останавливается до тех пор, пока не будет найдено хотя бы одно разбиение в узле, даже если	int float or auto, sqrt log2	default=None

Параметр	Описание	Возможные значения	Значение по умолчанию
	заданные условия не выполняются.		
random_state	Если int, random_state - это начальное число, используемое генератором случайных чисел; Если RandomState, то random_state является генератором случайных чисел; Если None, генератор случайных чисел является экземпляром RandomState, используемым np.random.	int or RandomState	default=None
max_leaf_ nodes	Дерево с количеством листов = max_leaf_nodes. Будут выбраны лучшие узлы по относительному уменьшению примесей. Если None, то неограниченное количество листов.	int	default=None
min_impurit y_decrease	Узел будет разбит, если это разбиение приведет у уменьшению примеси, больше или равное этому значению. Взвешенное уравнение уменьшения примеси имеет следующий вид: N_t / N * (impurity - N_t_R / N_t * right_impurity - N_t_L / N_t * left_impurity) где N - общее количество объектов, N_t - количество объектов в текущем узле, N_t_L - количество объектов в левом дочернем элементе, а N_t_R - количество объектов в правом дочернем элементе. N, N_t, N_t_R и N_t_L все ссылаются на взвешенную сумму, если параметр sample_weight определен.	float	default=0.0
min_impurity _split	Порог для ранней остановки роста деревьев. Узел разветвляется, если его примесь	float	default=1e-7

Параметр	Описание	Возможные значения	Значение по умолчанию
	выше указанного порога, в противном случае это лист. Устаревший параметр, min_impurity_split устарел в пользу min_impurity_decrease, рекомендовано использовать min_impurity_decrease.		
class_weight	Веса, связанные с классами в форме {class_label: weight}. Если «None», все классы должны иметь вес один. Для задач с несколькими выходами используется list of dict который может быть предоставлен в том же порядке, что и столбцы y. Обратите внимание, что для нескольких выходов (включая несколько меток) веса должны быть определены для каждого класса каждого столбца в его собственном словаре. Например, для четырехклассной классификации весовые коэффициенты должны быть [{0: 1, 1: 1}, {0: 1, 1: 5}, {0: 1, 1: 1}, {0: 1, 1: 1}] вместо [{1: 1}, {2: 5}, {3: 1}, {4: 1}]. В «balanced» режиме значения y используются для автоматической регулировки весов, обратно пропорциональных частотам классов во входных данных как n_samples / (n_classes * np.bincount (y)) Для мульти-вывода веса каждого столбца y будут умножены. Обратите внимание, что эти веса будут умножены на sample_weight (переданный через метод fit), если указан sample_weight.	dict list of dict or balanced	default=None

Параметр	Описание	Возможные значения	Значение по умолчанию
presort	предварительно отсортировано, по умолчанию = «deprecated» Этот параметр устарел.	deprecated	default=deprecated
ccp_alpha	Параметр сложности, используемый для сокращения минимальной стоимости и сложности. Будет выбрано поддерево с наибольшей стоимостью затрат, которое меньше, чем ccp_alpha. По умолчанию отсечение ветвей не выполняется.	non-negative float	default=0.0

Таблица 1.2. Атрибуты классификатора Classification and Regression Trees

Атрибут	Описание	Тип
classes	Метки классов (с одним выходом) или список массивов меток классов (с несколькими выходами).	ndarray размера n_classes or list of ndarray
feature_importances	значения атрибутов	ndarray размера n_features
max_features	Предполагаемое значение max_features. Количество объектов, которые следует учитывать при поиске лучшего разделения	int
n_classes	Количество классов (для задач с одним выходом) или список, содержащий количество классов для каждого выхода (для задач с несколькими выходами)	int or list of int
n_features	Количество входных атрибутов при обучении	int
n_outputs	Количество выходных атрибутов при обучении	int
tree	Базовый объект дерево решений	Tree

Таблица 1.3. Методы классификатора Classification and Regression Trees

Метод	Описание
apply(self, X[, check_input])	Возвращает индекс листа, в котором предсказан каждый образец.
cost_complexity_pruning_**path**(self, X, y[, …])	Вычисляет путь сокращения во время минимального сокращения стоимости сложности.
decision_path(self, X[, check_input])	Возвращает путь решения в дереве.
fit(self, X, y[, sample_weight, …])	Обучает классификатор - дерево решений на обучающем наборе (X, Y).
get_depth(self)	Возвращает глубину дерева решений
get_n_leaves(self)	Возвращает количество листьев дерева решений
get_params(self[, deep])	Получить параметры для оценочной функции
predict(self, X[, check_input])	Предсказывает класс или значение регрессии для X.
predict_log_proba(self, X)	Прогнозировать логарифм вероятности класса входных объектов X.
predict_proba(self, X[, check_input])	Предсказывает класс вероятностей входных выборок X
score(self, X, y[, sample_weight])	Возвращает среднюю точность данных и данных теста.
set_params(self, **params)	Установка параметров функции оценки.

1.11. Реализация классификатора в Scikit-Learn

Первый шаг в реализации классификатора – его импорт в Python.

```
from sklearn.tree import DecisionTreeClassifier
```

После этого нужно создать экземпляр классификатора. Сделать это можно создав переменную и вызвав функцию, связанную с классификатором.

```
tree_1=DecisionTreeClassifier()
```

Теперь классификатор нужно обучить.

```
tree_1.fit(X_train, Y_train)
```

После обучения модели уже можно подавать в классификатор данные для решения задачи классификации или регрессии.

```
Y_test_predict=tree_1.predict(X_test)
```

Далее можно выполнить оценку классификатора. Точность классификации измерять проще всего, и поэтому этот параметр чаще всего используется. Значение точности – это число правильных прогнозов, делённое на число всех прогнозов или, проще говоря, отношение правильных прогнозов ко всем.

```
mean_accuracy = tree_1.score(X_test, Y_test)
```

Пример построения классификатора на примере задачи классификации сортов ирисов приведен в приложении А и в файле 1_Scikit_learn_DecisionTreeClassifier_iris.ipynb.

1.12. Порядок выполнения работы

1. Выбрать одну из баз данных (согласовать с преподавателем), список которых представлен на сайте http://archive.ics.uci.edu/ml/. Скачать данные с сайта.

2. Исходные данные должны храниться в файле. Программным способом считать эти данные и сформировать обучающую, проверочную и тестовую выборки. Обучающая выборка должна включать достаточное количество примеров для обучения. Тестовая выборка может быть вполовину меньше обучающей. Примеры из тестовой выборки не должны быть включены в обучающую. Для оценки обучения модели можно использовать проверочную (валидационную) выборку (при достаточном количестве исходных данных) или же использовать кросс-валидацию (если данных мало).

3. Разработать программу на языке Python с использованием библиотек Pandas, Scikit-Learn. Программа должна обеспечивать: построение дерева (правил) алгоритмом согласно варианту на обучающей выборке; для деревьев решений реализовать механизм упрощения деревьев; тестирование дерева (правил); вычисление точности и ошибки классификации; сохранение / считывание модели (деревья или правила); вывод на экран структуры дерева, можно в виде правил.

4. Согласно варианту заданий (таблица 1.4) построить классификатор таким образом, чтобы точность на обучающей выборке была не 100%, но более

чем 90% (желательно около 97-98%). В процессе построения классификатора, записывайте в таблицу, какие у вас получались точности для различных моделей на обучающей и тестовой выборках в процессе подбора гиперпараметров.

Таблица 1.4. Варианты заданий

Вариант	Метод	Вариант	Метод
1	ID3	13	ID3
2	CART (классификация)	14	CART(Регрессия)
3	Naive Bayes	15	Naive Bayes
4	Random Forests	16	Random Forests
5	CART (Регрессия)	17	CART (Регрессия)
6	Naive Bayes	18	Naive Bayes
7	ID3	19	ID3
8	CART (классификация)	20	CART (классификация)
9	Random Forests	21	Random Forests
10	ID3	22	ID3
11	Naive Bayes	23	Naive Bayes
12	Random Forests	24	Random Forests

1.13. Содержание отчета

1. Титульный лист.
2. Описание предметной области и постановка задачи.
3. Алгоритм по варианту.
4. Листинг программы.
5. Результаты работы программы:
   - общий вид обучающей, проверочной и тестовой выборок;
   - дерево, ансамбль деревьев, правила для классификации для классификации;
   - таблица экспериментов подбора гиперпараметров;
   - точность классификации итоговой модели на обучающей и тестовой выборках.

Лабораторная работа №2

Тема: применение библиотеки Scikit-Learn для решения задач кластеризации.

Цель: изучение алгоритмов кластеризации, изучение возможностей библиотеки Scikit-Learn для решения задач кластеризации.

2.1. Введение в кластеризацию

Кластеризация − группировка объектов на основе близости их свойств; каждый кластер состоит из схожих объектов, а объекты разных кластеров существенно отличаются.

Кластер − группа похожих объектов.

При решении задачи кластеризации:

− оптимальное количество кластеров в общем случае неизвестно.

− выбор меры «похожести» или близости свойств объектов между собой, как и критерия качества кластеризации, часто носит субъективный характер.

Кластеризацию используют, когда отсутствуют априорные сведения относительно классов, к которым можно отнести объекты исследуемого набора данных, либо когда число объектов велико, что затрудняет их ручной анализ.

Кластеризация отличается от классификации тем, что для проведения анализа не требуется иметь выделенную целевую переменную, с этой точки зрения она относится к классу unsupervised learning. Эта задача решается на начальных этапах исследования, когда о данных мало что известно. Ее решение помогает лучше понять данные, и с этой точки зрения задача кластеризации является описательной задачей (descriptive).

Большое достоинство кластерного анализа в том, что он позволяет производить разбиение объектов не по одному параметру, а по целому набору признаков.

2.2. Постановка задачи кластеризации

Дано − набор данных со следующими свойствами:

- каждый экземпляр данных выражается четким числовым значением;
- класс для каждого конкретного экземпляра данных неизвестен.

Найти:

- способ сравнения данных между собой (меру сходства);
- способ кластеризации;
- разбиение данных по кластерам.

Формально задача кластеризации описывается следующим образом.

Дано множество объектов данных I, каждый из которых представлен набором атрибутов.

Множество I определим следующим образом:

$$I = \{i_1, i_2, ..., i_3, ..., i_n\}, \quad (2.1)$$

где i_j − исследуемый объект.

Каждый из объектов характеризуется набором параметров:

$$i_j = \{x_1, x_2, ..., x_h, ..., x_m\}. \quad (2.2)$$

Требуется построить множество кластеров C и отображение *F* множества I на множество C, т. е. *F: I* → C. Отображение *F* задает модель данных, являющуюся решением задачи. Качество решения задачи определяется количеством верно классифицированных объектов данных.

Каждая переменная x_h может принимать значения из некоторого множества:

$$x_h = \{v_h^{\,1}, v_h^{\,2}, ...\}. \quad (2.3)$$

В данном примере значениями являются действительные числа.

Задача кластеризации состоит в построении множества:

$$C = \{c_1, c_2, ..., c_k, ...c_g\}. \quad (2.4)$$

Здесь c_k − кластер, содержащий похожие друг на друга объекты из множества *I*:

$$c_k = \{i_l, i_p \mid i_j \in I, i_p \in I \; u \; d(i_l, i_p) < \sigma\}, \qquad (2.5)$$

где σ– величина, определяющая меру близости для включения объектов в один кластер; $d(i_l, i_p)$ – мера близости между объектами, называемая расстоянием.

Неотрицательное значение $d(i_l, i_p)$ называется расстоянием между элементами i_l и i_p, если выполняются следующие условия:

а) $d(i_l, i_p) \geq 0$, для всех i_j и i_p;

б) $d(i_l, i_p) = 0$, тогда и только тогда, когда $i_j = i_p$;

в) $d(i_l, i_p) = d(i_p, i_l)$;

г) $d(i_l, i_p) \leq d(i_l, i_r) + d(i_r, i_p)$.

Если расстояние $d(i_l, i_p)$ меньше некоторого значения σ, то говорят, что элементы близки и помещаются в один кластер. В противном случае говорят, что элементы отличны друг от друга и их помещают в разные кластеры.

Большинство популярных алгоритмов, решающих задачу кластеризации, используют в качестве формата входных данных матрицу отличия *D*. Строки и столбцы матрицы соответствуют элементам множества *I*. Элементами матрицы являются значения $d(i_l, i_p)$ в строке j и столбце *p*. Очевидно, что на главной диагонали значения будут равны нулю:

$$D = \begin{pmatrix} 0 & d(e_1, e_2) & d(e_1, e_n) \\ d(e_2, e_1) & 0 & d(e_2, e_n) \\ d(e_n, e_1) & d(e_n, e_2) & 0 \end{pmatrix}. \qquad (2.6)$$

Большинство алгоритмов работают с симметричными матрицами. Если матрица несимметрична, то ее можно привести к симметричному виду путем следующего преобразования:

$$(D + D^m)/2. \qquad (2.7)$$

2.3. Меры близости, основанные на расстояниях, используемые в алгоритмах кластеризации

Расстояния между объектами предполагают их представление в виде точек m-мерного пространства R^m. В этом случае могут быть использованы различные подходы к вычислению расстояний.

Рассмотренные ниже меры определяют расстояния между двумя точками, принадлежащими пространству входных переменных. Используются следующие обозначения:

- $X_Q \subseteq R^m$ – множество данных, являющееся подмножеством m-мерного вещественного пространства;
- $x_i = (x_{i1},...,x_{im}) \in X_Q, i = \overline{1,Q}$ – элементы множества данных;
- $\bar{x} = \frac{1}{Q}\sum_{i=1}^{Q} x_i$ – среднее значение точек данных;
- $S = \frac{1}{Q-1}\sum_{i=1}^{Q}(x_i - \bar{x})(x_i - \bar{x})'$ – ковариационная матрица (*m*x*m).*

Итак, приведем наиболее известные меры близости.

Евклидово расстояние. Иногда может возникнуть желание возвести в квадрат стандартное евклидово расстояние, чтобы придать большие веса более отдаленным друг от друга объектам. Это расстояние вычисляется следующим образом.

$$d_2(x_i, x_j) = \sqrt{\sum_{t=1}^{m}\left(x_{it} - x_{jt}\right)^2} \qquad (2.8)$$

Расстояние по Хеммингу. Это расстояние является просто средним разностей по координатам. В большинстве случаев данная мера расстояния приводит к таким же результатам, как и для обычного расстояния Евклида, однако для нее влияние отдельных больших разностей (выбросов) уменьшается (т. к. они не возводятся в квадрат). Расстояние по Хеммингу вычисляется по формуле:

$$d_H(x_i, x_j) = \sum_{t=1}^{m} | x_{it} - x_{jt} | \qquad (2.9)$$

Расстояние Чебышева. Это расстояние может оказаться полезным, когда желают определить два объекта как «различные», если они различаются по какой-либо одной координате (каким-либо одним измерением). Расстояние Чебышева вычисляется по формуле.

$$d(x_i, x_j) = \max_{1 \le t \le m} | x_{it} - x_{jt} | \qquad (2.10)$$

Расстояние Махаланобиса преодолевает этот недостаток, но данная мера расстояния плохо работает, если ковариационная матрица высчитывается на всем множестве входных данных. В то же время, будучи сосредоточенной на конкретном классе (группе данных), данная мера расстояния показывает хорошие результаты:

$$d_M(x_l, x_j) = (x_l - x_j) S^{-1} (x_l - x_j)' \qquad (2.11)$$

Пиковое расстояние предполагает независимость между случайными переменными, что говорит о расстоянии в ортогональном пространстве. Но в практических приложениях эти переменные не являются независимыми:

$$d_L(x_i, x_j) = \frac{1}{m} \sum_{l=1}^{m} \frac{|x_{it} - x_{jt}|}{x_{it} + x_{jt}} \qquad (2.12)$$

Любую из приведенных мер расстояния можно выбирать с уверенностью лишь в том случае, если имеется информация о характере данных, подвергаемых кластеризации.

Так, например, пиковое расстояние предполагает независимость между случайными переменными, что говорит о расстоянии в ортогональном пространстве. Но в практических приложениях эти переменные не являются независимыми.

2.4. Базовые алгоритмы кластеризации

2.4.1. Классификация алгоритмов

Число методов разбиения множества на кластеры довольно велико. Все их можно подразделить на иерархические и неиерархические.

В неиерархических алгоритмах характер их работы и условие остановки необходимо заранее регламентировать.

В иерархических алгоритмах фактически отказываются от определения числа кластеров, строя полное дерево вложенных кластеров. Число кластеров определяется из предположений, в принципе, не относящихся к работе алгоритмов, например по динамике изменения порога расщепления (слияния) кластеров. Иерархические алгоритмы делятся:

а) на агломеративные, характеризуемые последовательным объединением исходных элементов и соответствующим уменьшением числа кластеров (построение кластеров снизу вверх);

б) на дивизимные (делимые), в которых число кластеров возрастает начиная с одного, в результате чего образуется последовательность расщепляющих групп (построение кластеров сверху вниз).

2.4.2. Иерархические агломеративные алгоритмы

На первом шаге все множество *I* представляется как множество кластеров:

$$c_1 = \{i_1\}, c_2 = \{i_2\}, \ldots, c_m = \{i_m\}. \qquad (2.13)$$

На следующем шаге выбираются два наиболее близких друг к другу (например, c_p и c_q) и объединяются в один общий кластер. Новое множество, состоящее уже из m - 1 кластеров, будет:

$$c_1 = \{i_1\}, c_2 = \{i_2\}, \ldots, c_p = \{i_p, i_q\}, \ldots, c_m = \{i_m\}. \qquad (2.14)$$

Повторяя процесс, получим последовательные множества кластеров, состоящие из (m - 2), (m - 3), (m - 4) и т. д.

В конце процедуры получится кластер, состоящий из т объектов и совпадающий с первоначальным множеством I.

На каждой итерации вместо пары самых близких кластеров образуется новый кластер. Расстояние от нового кластера до любого другого кластера вычисляется по расстояниям кластеров, которые образуют новый кластер. Эти расстояния к этому моменту уже должны быть известны. Для определения расстояния между кластерами можно выбрать разные способы. В зависимости от этого получают алгоритмы с различными свойствами. Существует несколько методов пересчета расстояний с использованием старых значений расстояний для объединяемых кластеров, отличающихся коэффициентами в формуле:

$$d_{rs} = \alpha_p d_{ps} + \alpha_q d_{qs} + \beta d_{pq} + \gamma \mid d_{ps} - d_{qs} \mid . \qquad (2.15)$$

Если кластеры p и q объединяются в кластер r и требуется рассчитать расстояние от нового кластера до кластера s, применение того или иного метода зависит от способа определения расстояния между кластерами, эти методы различаются значениями коэффициентов α_p, α_q, β и γ.

В табл. 2.1 приведены коэффициенты пересчета расстояний между кластерами α_p, α_q, β и γ.

Таблица 2.1. Иерархические агломеративные алгоритмы

Название метода	α_p	α_q	β	γ
Расстояние между ближайшими соседями − ближайшими объектами кластеров (Nearest neighbor)	1/2	1/2	0	- 1/2
Расстояние между самыми далекими соседями (Furthest neighbor)	1/2	1/2	0	1/2
Метод медиан − тот же центроидный метод, но центр объединенного кластера вычисляется как среднее всех объектов (Median clustering)	1/2	1/2	1/4	0
Среднее расстояние между кластерами (Between-groups linkage)	1/2	1/2	0	0
Среднее расстояние между всеми объектами пары кластеров с учетом расстояний внутри кластеров (Within-groups linkage)	$\frac{k_p}{k_p + k_q}$	$\frac{k_q}{k_p + k_q}$	0	0
Расстояние между центрами кластеров (Centroid clustering), или центроидный метод. Недостатком этого метода является то, что центр объединенного кластера вычисляется как среднее центров объединяемых кластеров, без учета их объема	$\frac{k_p}{k_p + k_q}$	$\frac{k_p}{k_p + k_q}$	$\frac{-k_p k_q}{k_p + k_q}$	0

Название метода	α_p	α_q	β	γ
Метод Уорда (Ward's method). В качестве расстояния между кластерами берется прирост суммы квадратов расстояний объектов до центров кластеров, получаемый в результате их объединения	$\frac{k_r + k_p}{k_r + k_p + k_q}$	$\frac{k_r + k_p}{k_r + k_p + k_q}$	$\frac{-k_r}{k_r + k_p + k_q}$	0

2.4.3. Иерархические дивизимные алгоритмы

Дивизимные кластерные алгоритмы, в отличие от агломеративных, на первом шаге представляют все множество элементов I как единственный кластер. На каждом шаге алгоритма один из существующих кластеров рекурсивно делится на два дочерних. Таким образом итерационно образуются кластеры сверху вниз. Такой подход применяют, когда необходимо разделить все множество объектов I на относительно небольшое количество кластеров.

Один из первых дивизимных алгоритмов был предложен Смитом Макнаотоном в 1965 году.

На первом шаге все элементы помещаются в один кластер C_1=I.

Затем выбирается элемент, у которого среднее значение расстояния от других элементов в этом кластере наибольшее. Среднее значение может быть вычислено, например, с помощью формулы:

$$D_{C1} = 1/N_{C1} \times \sum\sum d(i_p, i_q) \forall i_p, i_q \in C. \qquad (2.16)$$

Выбранный элемент удаляется из кластера C_1 и формирует первый член второго кластера C_2.

На каждом последующем шаге элемент в кластере C_1 для которого разница между средним расстоянием до элементов, находящихся в C_2, и средним расстоянием до элементов, остающихся в C_1, наибольшая, переносится в C_2.

Переносы элементов из C_1 в C_2 продолжаются до тех пор, пока соответствующие разницы средних не станут отрицательными, т.е. пока существуют, элементы, расположенные к элементам кластера C_2 ближе чем к элементам кластера C_1.

В результате один кластер делится на два дочерних, один из которых расщепляется на следующем уровне иерархии. Каждый последующий уровень применяет процедуру разделения к одному из кластеров, полученных на предыдущем уровне. Выбор расщепляемого кластера может выполняться по-разному.

В 1990 г. Кауфман и Роузеув предложили выбирать на каждом уровне кластер, для расщепления с наибольшим диаметром, который вычисляется по формуле:

$$D_C = max\, d(i_p, i_q) \forall i_p, i_q \in C. \qquad (2.17)$$

Рекурсивное разделение кластеров продолжается, пока все кластеры или не станут сиглетонами (т.е. состоящими из одного объекта), или пока все члены одного кластера не будут иметь нулевое отличие друг от друга.

2.4.4. Неиерархические алгоритмы

Большую популярность при решении задач кластеризации приобрели алгоритмы, основанные на поиске оптимального в определенном смысле разбиения множества данных на кластеры (группы). Во многих задачах в силу своих достоинств используются именно алгоритмы построения разбиения. Данные алгоритмы пытаются сгруппировать данные (в кластеры) таким образом, чтобы целевая функция алгоритма разбиения достигала экстремума (минимума). Рассмотрим основные алгоритмы кластеризации, основанных на методах разбиения. В данных алгоритмах используются следующие базовые понятия:

− обучающее множество (входное множество данных) M, на котором строится разбиение;

− метрика расстояния:

$$d_A^2(m_j, c^{(l)}) = \left\| m_j - c^{(l)} \right\|_A^2 = \left(m_j - c^{(l)}\right)' A \left(m_j - c^{(l)}\right), \qquad (2.18)$$

где матрица A определяет способ вычисления расстояния. Например, для единичной матрицы будем использовать расстояние по Евклиду;

− вектор центров кластеров C;

- матрица разбиения по кластерам U;
- целевая функция J =J(M, d, C, U);
- набор ограничений.

2.5. Алгоритм k-means

Вначале выбирается *k* произвольных исходных центров − точек в пространстве всех объектов. Не очень критично, какие именно это будут центры, процедура выбора исходных точек отразится, главным образом, только на времени счета. Например, это могут быть первые *k* объектов множества *I.*

Дальше итерационно выполняется операция двух шагов.

На первом шаге все объекты разбиваются на *k* групп, наиболее близких к одному из центров. Близость определяется расстоянием, которое вычисляется одним из описанных ранее способов (например, берется Евклидово расстояние).

На втором шаге вычисляются новые центры кластеров. Центры можно вычислить как средние значения переменных объектов, отнесенных к сформированным группам. Новые центры, естественно, могут отличаться от предыдущих. Естественно, что некоторые точки, ранее относящиеся к одному кластеру, при новом разбиении попадают в другой.

Рассмотренная операция повторяется рекурсивно до тех пор, пока центры кластеров (соответственно, и границы между ними) не перестанут меняться.

2.6. Обзор методов кластеризации с помощью Python и Scikit-Learn

Кластеризация в Scikit-Learn представлена группой методов, с которыми можно ознакомится в документации к библиотеке https://scikit-learn.org/stable/modules/clustering.html.

Модели реализованы в виде классов. Для примера рассмотрим модель k-means. В таблице 2.2 представлены параметры этого класса. В таблице 2.3 представлены атрибуты этого класса. А в таблице 2.4 методы этого класса.

Более подробное описание KMeans в https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

Таблица 2.2. Параметры KMeans

Параметр	Описание	Возможные значения	Значение по умолчанию
n_clusters	Количество кластеров для формирования.	int	8
init	Методы для инициализации	‘k-means++’ , ‘random’, ndarray, callable	‘k-means++’
n_init	Количество раз, когда алгоритм k-средних будет выполняться с различными центроидами. Конечными результатами будут наилучшие результаты последовательных запусков n_init с точки зрения инерции.	int	10
max_iter	Максимальное число итераций алгоритма k-средних для одного прогона.	int	300
tol	Относительная толерантность по отношению к норме Фробениуса разности центров кластеров двух последовательных итераций для объявления сходимости.	float	1e-4
precompute_distances	Предварительный расчет расстояний (быстрее, но занимает больше памяти).	‘auto’, True, False	auto
verbose	Режим детализации	int	0
random_state	Определяет генерацию случайных чисел для инициализации центроида. Используйте int, чтобы сделать случайность детерминированной	int	None
copy_x	При предварительном вычислении расстояний данные центрируются более точно с числовой точки зрения. Если copy_x имеет значение True, то исходные данные не изменяются. Если значение False, исходные данные изменяются и помещаются обратно до возвращения функции, но небольшие числовые различия	bool	True

Параметр	Описание	Возможные значения	Значение по умолчанию
	могут быть введены путем вычитания, а затем добавления среднего значения данных.		
n_jobs	Количество потоков OpenMP, используемых для вычисления. Параллелизм осуществляется выборочно на главном цикле, который присваивает каждому образцу его ближайший центр.	int	None
algorithm	K-средних алгоритм использования. Классический алгоритм EM-стиля является “полным”. Вариация “elkan” более эффективна для данных с четко определенными кластерами, однако он использует больше в памяти из-за выделения дополнительного массива фигур (n_samples, n_clusters).	“auto”, “full”, “elkan”	“auto”

Таблица 2.3. Атрибуты KMeans

Атрибут	Описание	Тип
cluster_centers_	Координаты центров скоплений.	ndarray of shape (n_clusters, n_features)
labels_	Метки каждой точки	ndarray of shape (n_samples,)
intertia_	Сумма квадратов расстояний до ближайшего центра кластера.	Float
n_iter_	Количество выполненых итераций.	int

Таблица 2.4. Методы KMeans

Метод	Описание
fit(X[, y, sample_weight])	Выполнение кластеризации.
fit_predict(X[, y, sample_weight])	Вычисление центров кластеров и прогнозирование кластерного индекса для каждого объекта выборки.

fit_transform(X[, y, sample_weight])	Выполнение кластеризации и преобразование X в пространство расстояний между кластерами.
get_params([deep])	Получение параметров модели.
predict(X[, sample_weight])	Предсказание ближайшего кластера, к которому принадлежит каждый образец в X.
score(X[, y, sample_weight])	Сопоставление значения X задачи K-средних.
set_params(**params)	Установка параметров мадели.
transform(X)	Преобразование X в пространство кластерного расстояния.

Рассмотрим так же модель DBSCAN. Ссылка на библиотеку DBSCAN https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Таблица 2.5. Параметры DBSCAN

Параметр	Описание	Возможные значения	Значение по умолчанию
eps	Максимальное расстояние между двумя образцами, при котором можно рассматривать их как находящееся в окрестности друг друга. Это не максимальная граница расстояний между точками внутри кластера. Это самый важный параметр DBSCAN, который необходимо выбрать соответствующим образом для вашего набора данных и функции расстояния.	float	0.5
min_samples	Число точек (или общий вес) в окрестности точки, которая должна рассматриваться как базовая точка.	float	0.5
metric	Метрика, используемая при вычислении расстояния между экземплярами в массиве объектов.	string, or callable	euclidean

Параметр	Описание	Возможные значения	Значение по умолчанию
metric_params	Дополнительные аргументы для функции метрики.	dict	None
algorithhm	Алгоритм, который будет использоваться модулем NearestNeighbors для вычисления расстояний и поиска ближайших соседей.	‘auto’, ‘ball_tree’, ‘kd_tree’, ‘brute’	auto
leaf_size	Размер листа, который передается в BallTree или cKDTree. Это может повлиять на скорость построения запроса, а также на объем памяти, необходимый для хранения дерева. Оптимальное значение зависит от характера задачи.	int	30
p	Степень метрики Минковского, которая будет использоваться для вычисления расстояния между точками.	float	None
n_jobs	Количество выполняемых параллельных заданий. None означает 1, если только в контексте joblib.parallel_backend. -1 означает использование всех процессоров.	int	None

Таблица 2.6. Атрибуты DBSCAN

Атрибут	Описание	Тип
core_sample_indices_	Индекс ядра образца.	ndarray of shape (n_core_samples,)
components_	Копия каждого образца ядра, найденного путем обучения.	ndarray of shape (n_core_samples, n_features)
labels_	Метки кластеров для каждой точки в наборе данных задаются функцией fit(). Шуму присваивается метка -1.	ndarray of shape (n_samples)

Таблица 2.7. Методы DBSCAN

Метод	Описание
fit(X[, y, sample_weight])	Выполнение кластеризации.
fit_predict(X[, y, sample_weight])	Выполнить кластеризацию DBSCAN от и вернуть метки кластера.
get_params([deep])	Получите параметры модели.
set_params(**params)	Установите параметры модели.

2.7. Реализация метода k-средних в Scikit-Learn

Рассмотрим набор данных ирисов Фишера. Датасет представляет набор из 150 записей с пятью атрибутами в следующем порядке: длина чашелистика (sepal length), ширина чашелистика (sepal width), длина лепестка (petal length), ширина лепестка (petal width) и класс, соответствующий одному из трех видов: Iris Setosa, Iris Versicolor или Iris Virginica, обозначенных соответственно 0, 1, 2. Для решения задач кластеризации данных мы используем Python, библиотеку scikit-learn для загрузки и обработки набора данных и matplotlib для визуализации. Ниже представлен программный код для исследования исходного набора данных.

```
# Импортируем библиотеки
from sklearn import datasets
import matplotlib.pyplot as plt
# Загружаем набор данных
iris_df = datasets.load_iris()
# Методы, доступные для набора данных
print(dir(iris_df))
# Признаки
print(iris_df.feature_names)
# Метки
print(iris_df.target)
# Имена меток
print(iris_df.target_names)
# Разделение набора данных
```

```
x_axis = iris_df.data[:, 0]  # Sepal Length
y_axis = iris_df.data[:, 1]  # Sepal Width
# Построение
plt.xlabel(iris_df.feature_names[0])
plt.ylabel(iris_df.feature_names[1])
plt.scatter(x_axis, y_axis, c=iris_df.target)
plt.show()
```

В результате запуска программы увидим текст и изображение.

```
['DESCR', 'data', 'feature_names', 'target',
'target_names']
['sepal length (cm)', 'sepal width (cm)', 'petal length
(cm)', 'petal width (cm)']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2]
['setosa' 'versicolor' 'virginica']
```

На диаграмме (рис. 2.1) фиолетовым цветом обозначен вид Setosa, зеленым – Versicolor и желтым – Virginica. При построении были взяты лишь два признака. Можно проанализировать как разделяются классы при других комбинациях параметров. Цель кластеризации данных состоит в том, чтобы выделить группы примеров с похожими чертами и определить соответствие примеров и кластеров. При этом исходно у нас нет примеров такого разбиения. Это аналогично тому, как если бы в приведенном наборе данных у нас не было меток, как на рисунке 2.2.

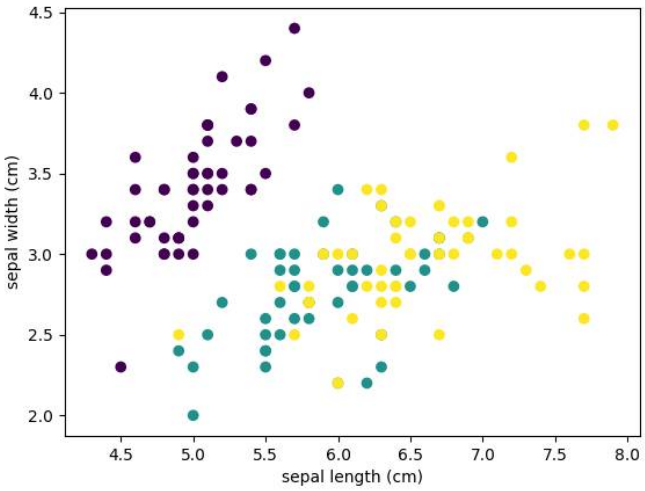


Рисунок 2.1. Визуализация данных

На диаграмме фиолетовым цветом обозначен вид Setosa, зеленым – Versicolor и желтым – Virginica. При построении были взяты лишь два признака. Можно проанализировать как разделяются классы при других комбинациях параметров. Цель кластеризации данных состоит в том, чтобы выделить группы примеров с похожими чертами и определить соответствие примеров и кластеров. При этом исходно у нас нет примеров такого разбиения. Это аналогично тому, как если бы в приведенном наборе данных у нас не было меток, как на рисунке ниже.

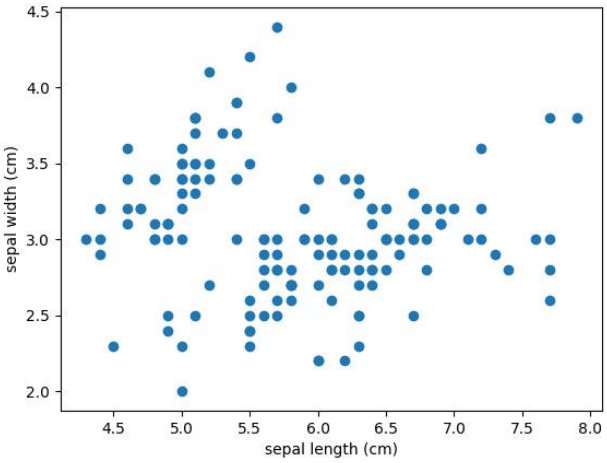


Рисунок 2.2. Данные

Наша задача – используя все имеющиеся данные, предсказать соответствие объектов выборки их классам, сформировав таким образом кластеры. K-means это итеративный алгоритм кластеризации, основанный на минимизации

суммарных квадратичных отклонений точек кластеров от центроидов (средних координат) этих кластеров.

Первоначально выбирается желаемое количество кластеров. Поскольку нам известно, что в нашем наборе данных есть 3 класса, установим параметр модели `n_clusters` равный трем.

Теперь случайным образом из входных данных выбираются три элемента выборки, в соответствие которым ставятся три центра кластера. Далее ищем ближайшего соседа текущего центроида. Добавляем точку к соответствующему кластеру и пересчитываем положение центроида с учетом координат новых точек. Алгоритм заканчивает работу, когда координаты каждого центроида перестают меняться. Центроид каждого кластера в результате представляет собой набор значений признаков, описывающих усредненные параметры объектов кластера.

```
# Импортируем библиотеки
from sklearn import datasets
from sklearn.cluster import KMeans
# Загружаем набор данных
iris_df = datasets.load_iris()
# Описываем модель
model = KMeans(n_clusters=3)
# Проводим моделирование
model.fit(iris_df.data)
# Предсказание на единичном примере
predicted_label = model.predict([[7.2, 3.5, 0.8, 1.6]])
# Предсказание на всем наборе данных
all_predictions = model.predict(iris_df.data)
# Выводим предсказания
print(predicted_label)
print(all_predictions)
```

Результат:

```
[1]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 2 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2 2 2 0 0
2 2 2 2 0 2 0 2 0 2 2 0 0 2 2 2 2 2 0 2 2 2 2 0 2 2 2 0 2
2 2 0 2 2 0]
```

При выводе данных нужно понимать, что алгоритм не знает ничего о нумерации классов, и числа 0, 1, 2 – это лишь номера кластеров, определенных в результате работы алгоритма. Так как исходные точки выбираются случайным образом, вывод будет несколько меняться от одного запуска к другому.

Характерной особенностью набора данных ирисов Фишера является то, что один класс (Setosa) легко отделяется от двух остальных. Это заметно и в приведенном примере.

2.8. Реализация иерархической кластеризации в Scikit-Learn

Иерархическая кластеризация, как следует из названия, представляет собой алгоритм, который строит иерархию кластеров. Этот алгоритм начинает работу с того, что каждому экземпляру данных сопоставляется свой собственный кластер. Затем два ближайших кластера объединяются в один и так далее, пока не будет образован один общий кластер.

Результат иерархической кластеризации может быть представлен с помощью дендрограммы. Рассмотрим этот тип кластеризации на примере данных для различных видов зерна.

```
# Импортируем библиотеки
from scipy.cluster.hierarchy import linkage, dendrogram
import matplotlib.pyplot as plt
import pandas as pd
# Создаем датафрейм
seeds_df = pd.read_csv("http://qps.ru/jNZUT")
```

```
# Исключаем информацию об образцах зерна, сохраняем для
дальнейшего использования
varieties = list(seeds_df.pop('grain_variety'))
# Извлекаем измерения как массив NumPy
samples = seeds_df.values
# Реализация иерархической кластеризации при помощи
функции linkage
mergings = linkage(samples, method='complete')
# Строим дендрограмму, указав параметры удобные для
отображения
dendrogram(mergings, labels=varieties, leaf_rotation=90,
leaf_font_size=6,  )
plt.show()
```

Можно видеть, что в результате иерархической кластеризации данных естественным образом произошло разбиение на три кластера, обозначенных на рисунке 2.3 различным цветом. При этом исходно число кластеров не задавалось.

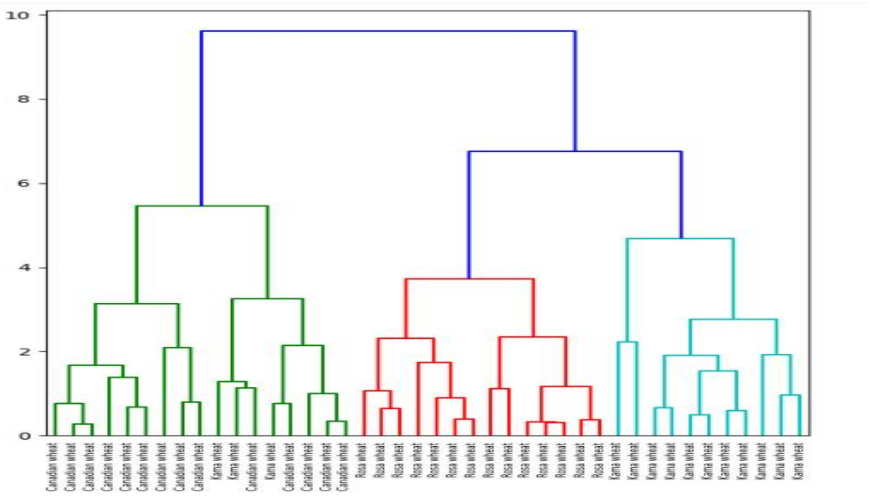


Рисунок 2.3 Иерархическая кластеризация данных

2.9. Реализация кластеризации на основе плотности DBSCAN

DBSCAN (Density-Based Spatial Clustering of Applications with Noise, плотностной алгоритм пространственной кластеризации с присутствием шума)

– популярный алгоритм кластеризации, используемый в анализе данных в качестве одной из замен метода k-средних.

Метод не требует предварительных предположений о числе кластеров, но нужно настроить два других параметра: eps и min_samples. Данные параметры – это соответственно максимальное расстояние между соседними точками и минимальное число точек в окрестности (количество соседей), когда можно говорить, что эти экземпляры данных образуют один кластер. В scikit-learn есть соответствующие значения параметров по умолчанию, но, как правило, их приходится настраивать самостоятельно.

```
# Импортируем библиотеки
from sklearn.datasets import load_iris
import matplotlib.pyplot as plt
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA
# Загружаем датасет
iris = load_iris()
# Определяем модель
dbscan = DBSCAN()
# Обучаем
dbscan.fit(iris.data)
# Уменьшаем размерность при помощи метода главных
компонент
pca = PCA(n_components=2).fit(iris.data)
pca_2d = pca.transform(iris.data)
# Строим в соответствии с тремя классами
for i in range(0, pca_2d.shape[0]):
    if dbscan.labels_[i] == 0:
        c1 = plt.scatter(pca_2d[i, 0], pca_2d[i, 1],
c='r', marker='+')
    elif dbscan.labels_[i] == 1:
```

```
        c2 = plt.scatter(pca_2d[i, 0], pca_2d[i, 1],
c='g', marker='o')
    elif dbscan.labels_[i] == -1:
        c3 = plt.scatter(pca_2d[i, 0], pca_2d[i, 1],
c='b', marker='*')
plt.legend([c1, c2, c3], ['Кластер 1', 'Кластер 2',
'Шум'])
plt.title('DBSCAN нашел 2 кластера и шум')
plt.show()
```

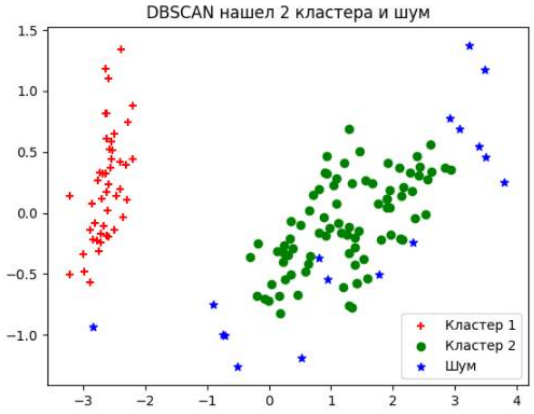


Рисунок 2.4. Визуализация метода DBSCAN с выделением шума

2.10. Порядок выполнения работы

1. Выбрать одну из баз данных (согласовать с преподавателем), список которых представлен на сайте http http://archive.ics.uci.edu/ml/datasets.php?format=&task=clu&att=&area=&numAtt=&numIns=&type=&sort=nameUp&view=table. Скачать данные с сайта или можно использовать данные, полученные на практике, результаты НИРС или данные для бакалаврской работы.

2. Исходные данные должны храниться в файле. Программным способом считать эти данные и сформировать обучающую и тестовую выборки.

3. Разработать программу на любом языке программирования с использованием любых существующих библиотек. Рекомендован язык Python с использованием библиотек NumPy, Pandas, Scikit-Learn. Программа должна

обеспечивать: реализацию метода кластеризации, который указан в задании; визуализацию результатов; тестирование модели; оценка точности модели; сохранение / считывание модели; вывод модели.

4. Согласно варианту заданий (таблица 2.8) построить несколько моделей. В процессе построения моделей, анализируйте ее качество и записывайте результаты в таблицу, указывая значения гиперпараметров.

Таблица 2.8. Варианты заданий

Вариант	Метод	Вариант	Метод
1	K-Means	13	Mean-shift
2	Affinity propagation	14	Spectral clustering
3	Mean-shift	15	FeatureAgglomeration
4	Spectral clustering	16	Agglomerative clustering
5	FeatureAgglomeration	17	DBSCAN
6	Agglomerative clustering	18	OPTICS
7	DBSCAN	19	Gaussian mixtures
8	OPTICS	20	Spectral clustering
9	Mean-shift	21	K-Means
10	Gaussian mixtures	22	Affinity propagation
11	K-Means	23	Mean-shift
12	Affinity propagation	24	Spectral clustering

2.11. Содержание отчета

1. Титульный лист.

2. Описание предметной области и постановка задачи.

3. Описание параметров модели, описать полученную модель (иметь представление, что она представляет собой).

4. Листинг программы.

5. Результаты работы программы: общий вид обучающей, проверочной и тестовой выборок; построенная модель; результаты тестирования; таблица экспериментов подбора гиперпараметров.

Лабораторная работа №3

Тема: нейронные сети для решения задач классификации и регрессии.

Цель работы: познакомиться с понятием полносвязной нейронной сети, с понятием метрик качества моделей обучения, функциями потерь и оптимизаторами обучения, с понятием «переобучение» модели нейронной сети; научиться строить с нуля и обучать нейронную сеть с помощью Keras и TensorFlow.

3.1. Базовые объекты и параметры объектов нейронных сетей в TensorFlow

Необходимые объекты и их параметры:

1. tensorflow.keras.models.Sequential – это базовая модель нейронной сети, которая, по сути, является контейнером для последовательно помещенных в нее слоев. Не имеет параметров;

2. tensorflow.keras.layers.Dense – это объект полносвязного слоя нейронной сети. Определяется следующими параметрами:

– units — это количество нейронов в данном слое;

– input_dim – размерность входного слоя (только для слоя, непосредственно следующего за входными данными, для последующих слоев входная размерность определяется автоматически исходя из размерности предыдущего слоя);

– activation – функция активации, которая будет использована для данного слоя («relu», «softmax» и др.).

Есть и другие параметры данного слоя.

3.2. Метрики качества

Метрики под каждую задачу могут выбираться индивидуально. Есть ряд общепринятых и часто употребляемых метрик качества.

Для задач регрессии чаще всего применяются:

– среднее квадратичное отклонение (mean square error – MSE):

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \tilde{y}_i)^2 \quad (3.1)$$

– среднее абсолютное отклонение (mean absolute error – MAE):

$$MAE = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i| \quad (3.2)$$

MSE – это среднее квадратов всех отклонений предсказанных значений от истинных. Квадрат нужен для того, чтобы избежать отрицательных значений. Таким образом, данная метрика тем лучше, чем ближе ее значение к нулю (отсюда диапазон значений данной метрики от 0 до +∞). У нее есть определенные недостатки. Из-за наличия квадрата она завышает значение больших ошибок (больше единицы) и преуменьшает значение ошибок, значение которых меньше единицы.

MAE – это среднее абсолютных значение всех отклонений. Выполняет те же функции, что и MSE, но является более интерпретируемой, так как берет ошибки по модулю, не выполняя нелинейного преобразования.

Существует еще масса модификаций этих метрик (RMSE, MAPE, MSLE), которые можно увидеть в документации TensorFlow https://www.tensorflow.org/api_docs/python/tf/keras/metrics

Для задач классификации существуют свои метрики. Рассмотрим их для простоты понимания на примере бинарной классификации, то есть классификации на два класса: отрицательный и положительный (класс 0 и класс 1):

– accuracy – полная точность:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (3.3)$$

где TP – это количество верно классифицированных меток положительного класса; TN – это количество верно классифицированных меток отрицательного класса; FP – это количество неверно классифицированных меток положительного класса; FN – это количество неверно классифицированных меток отрицательного класса.

– precision – точность для положительного класса:

$$precision = \frac{TP}{TP + FP} \tag{3.4}$$

– recall:

$$recall = \frac{TP}{TP + FN} \tag{3.5}$$

– Cross Entropy:

$$-plog\,(a_i) - (1 - p)\,log\,(1 - a_i) \tag{3.6}$$

где p – настоящая вероятность класса (это наша ручная разметка, и она имеет значения либо 0, либо 1, так как мы, в отличие от нейронной сети, точно уверены в значениях классов); a_i – вероятность класса i-го объекта, предсказанная нейронной сетью; log – натуральный логарифм числа.

Accuracy – это точность в целом по выборке. Просто считаются все верные ответы по всем классам, и эта сумма делится на общее количество классов. Это одна из часто применяемых метрик, но все же она имеет свой недостаток. Например, класс 1 – «клиент не откажется от услуг», а 0 – «откажется». Обычно в задачах подобного рода мы имеем несбалансированную выборку. Предположим, что есть 100 клиентов, из них 90 отказались от услуг, а 10 – нет. Таким образом, если нейронная сеть будет увеличивать accuracy, то ей достаточно прогнозировать для всех клиентов, что они откажутся от услуг, и тогда она получит качество (0(TP) + 90(TN)) / 100 = 0,90 или 90 %. Получается, что в случае дисбаланса классов применять данную метрику опасно.

Precision – это также точность, но как видно по формуле, она следит лишь за положительным классом. И в ситуации с предыдущим примером даст честные 0(TP) / (0(TP) + 0(FP)) = 0 % (0/0 на самом деле неопределенность, но функция в TensorFlow обработает это исключение). Но у данной метрики также есть недостаток. Предположим, что на той же задаче модель верно определила 5 клиентов, которые не откажутся от услуги (класс 1), а всех остальных отнесла в класс 0. Тогда 5(TP) / (5(TP) + 0(FP)) = 1 или 100 %. Данную метрику интересуют доля верных ответов среди всех ответов, отнесенных к 1-му классу, она следит,

чтобы не было ложно положительных ответов (то есть истинно нулевых, отнесенных к 1-му классу).

Recall – это полнота. Она смотрит уже на количество верных ответов для положительного класса. В предыдущей ситуации полнота даст нам 5(TP) / (5(TP) + 5(FN)) = 0,5 или 50 %. Ведь мы верно угадали лишь 50 % из всех правильных ответов. Но можно заметить, что данная метрика будет вводить нас в заблуждение в другой ситуации. Если модель предскажет всех как положительный класс, то получим 10(TP) / (10(TP) + 0(FN)) = 1 или 100 %. Но при этом мы захватили в положительный класс 90 реально ушедших клиентов и даже этого не заметили. С это ситуацией как раз боролась метрика presision.

Итак presision и recall – это две стороны одной медали и в реальных задачах нужно балансировать между ними, исходя из специфики задачи. К тому же у них есть общий недостаток – они применимы лишь для бинарной классификации. А вот accuracy легко применима и для многоклассовой.

Так же стоит отметить, что метрики accuracy, precision, recall имеют диапазон значений от 0 (если все неверно) до 1 (если все верно).

Cross Entropy – данная метрика применима как для бинарной, так и для многоклассовой классификации. На несбалансированном классе ведет себя нормально, так как она дает большой штраф как за неверную классификацию в сторону отрицательного класса, так и в сторону положительного. Допустим, класс должен быть положительным, тогда p = 1, а нейронная сеть предсказала достаточно уверенно (с вероятностью a = 0,01), что класс отрицательный. Тогда -1*log(0,01) – (1-1)*log(1-0,01) = -1*(-4,605) – (0)*−0,01 = 4,605. И это только на одном примере. А если вспомнить наш пример с отказом клиентов от услуг, где модель начинает всем предсказывать, что они откажутся от услуги, то есть кидать их в отрицательный класс 0, а на самом деле 10 человек должны быть в положительном, то получим метрику 10*4,605 = 46,05, что весьма и весьма плохо. Из этих же выкладок следует, что диапазон данной метрики от 0 до +∞, где чем ближе к 0, тем лучше.

Как и в случае с регрессией, это не все возможные метрики. Про остальные можно почитать по ссылке на документацию TensorFlow, предоставленной выше.

3.3. Функции потерь и оптимизаторы обучения

Нейронной сети, чтобы обучаться, также нужна какая-то мера. Она должна уметь сравнить свой ответ с верным ответом из учебного ответа, оценить, насколько ее ответ близок к истине или далек от нее, а затем с помощью специального алгоритма обратного распространения ошибки обновить все свои веса. Эта функция должна быть дифференцируемой, чтобы в дальнейшем нейронная сеть смогла посчитать производные по этой функции и определить градиент данной функции. Так как данная функция определяет, насколько нейронная сеть ошиблась, то ее принято называть функцией потерь (loss function). Из вышеперечисленных метрик функциями потерь могут послужить MAE, MSE, Cross Entropy и другие метрики, не упомянутые в курсе, но удовлетворяющие требованию дифференцируемости. Полный список функций потерь можно увидеть в документации TensorFlow https://www.tensorflow.org/api_docs/python/tf/keras/losses.

За сам процесс распространения посчитанной ошибки обратно на все веса модели нейронной сети отвечают так называемые оптимизаторы. Это функции, которые распространяют ошибку, используя различные стратегии:

- SGD – стохастический градиентный спуск;
- RMSprop – модификация градиентного спуска;
- Adam – модификация градиентного спуска.

Полный список данных функций можно увидеть в документации TensorFlow https://www.tensorflow.org/api_docs/python/tf/keras/optimizers.

3.4. Пример построение простой полносвязной нейронной сети с помощью Keras и TensorFlow

Импортируем необходимые объекты:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

Создаем объект последовательной модели нейронной сети:

```
model = Sequential()
```

Помещаем необходимые слои в модель. Например, добавим слой обрабатывающий входные данные размерностью 784 элемента. Количество нейронов в слое зададим 700, а функцию активации relu.

```
model.add(Dense(units = 700, input_dim = 784, activation = "relu"))
```

И пусть выходной слой, выдающий вероятность принадлежности к одному из 5 классов с функцией активации *softmax*.

```
model.add(Dense(5, activation="softmax"))
```

Чтобы модель построилась и была готова к использованию, ее необходимо скомпилировать.

```
model.compile(loss = "categorical_crossentropy", optimizer = "SGD", metrics = ["accuracy"])
```

И вот тут мы видим три понятия: metrics (метрика, на основе которой мы делаем вывод о качестве модели), loss (функция потерь, на основе которой нейронная сеть понимает на сколько она ошиблась), optimizer (функция, которая реализует определенную стратегию обратного распространения ошибки по весам нейронной сети).

Затем нужно обучить нейронную сеть:

```
model.fit(x_train, y_train, batch_size=200, epochs=10, verbose=1)
```

где *x_train* – матрица, в которой каждая строка это отдельный объект, а каждый столбец – это отдельный признак объекта; *y_train* – вектор меток объектов (его длина должна совпадать с количеством строк в матрице); *batch_size* – количество объектов из матрицы *x_train*, которые делают проход по нейронной сети за один раз и после которых выполняется обновление весов нейронной сети; *epochs* – количество эпох, на протяжении которых будет

учиться нейронная сеть. Одна эпоха – это полный проход всех учебных данных через нейронную сеть. То есть к концу первой эпохи нейронная сеть уже увидит все обучающие данные и в последующих эпохах будет видеть их же, но в другом сочетании внутри одного *batch*. Последующие эпохи могут помочь сделать результат лучше; *verbose* – отвечает за выведение информации о процессе обучения. Если значение «0», то информация выводиться не будет, если «1», то мы увидим эту информацию по всем эпохам обучения.

По окончании обучения можно применять модель нейронной сети для предсказаний:

```
predictions = model.predict(x_to_predict)
```

Теперь в переменной predictions будут храниться результаты предсказаний по всем примерам, которые были поданы в матрице *x_to_predict*.

Также модель можно сохранить:

```
model.save('model')
```

И в дальнейшем использовать без предварительного обучения

```
loaded_model = tf.keras.models.load_model('model')
```

3.5. Оценка реального качества модели нейронной сети и наборы данных

И на данный момент оценить ее качество мы могли лишь по тем данным, которые нейронная сеть выводила нам в процессе обучения.

Epoch 49/50 300/300 [=====] — 1s 4ms/step — loss: 0.0661 — accuracy: 0.9761

Epoch 50/50 300/300 [=====] — 1s 4ms/step — loss: 0.0613 —;accuracy: 0.9782

Для реальной оценки качества модели выделяют отдельный набор данных – тестовый набор. К тестовому набору данных предъявляются определенные требования:

− имеет такую же структуру признаков, как и тренировочный набор;

− также имеет выходные метки;

− перед тем, как подавать его на вход нейронной сети, с ним необходимо выполнить всю последовательность преобразований, которые были применены к тренировочному набору;

− в процессе обучения нейронная сеть не должна «видеть» наблюдения из данного набора данных.

После того, как модель полностью обучилась, мы можем применить тестовый набор для реальной оценки качества следующим образом:

```
model.evaluate(x_test, y_test)
```

где `model` – это наша уже обученная модель, `evaluate` – это метод модели, который позволяет произвести оценку модели по ранее выбранной нами метрике. В данной задаче мы использовали `accuracy`.

Метод `evaluate` принимает два параметра: входные признаки `(x_test)` и метки верных ответов `(y_test.` Как было сказано выше, это точно такие наборы данных и меток, которые использовались при обучении, но которые нейронная сеть не видела в процессе обучения. Метод проверки выведет нам результат проверки в следующем формате:

313/313 [==============] — 1s 3ms/step —loss: 0.4385 — accuracy: 0.8988

Теперь мы можем провести сравнительный анализ качества модели оцененного на тренировочных данных и на тестовых (табл. 3.1).

Таблица 3.1 Сравнительный анализ качества модели по оценкам на тренировочном наборе данных и на тестовом.

Набор данных, использованный для проверки	Ошибка модели (loss)	Качество модели (accuracy)
На тренировочном наборе	0,0613	0,9782
На тестовом наборе	0,4385	0,8988

Как видно из таблицы, и значение ошибки, и значение метрики качества очень сильно расходится. Так как ошибка (loss) не нормирована в пределах 0 и 1, то ее несколько сложнее использовать для оценки расхождения показаний моделей. Поэтому будем ориентироваться на метрику качества. Ее легко можно перевести в процентное выражение и получить разность оценок качества моделей 0,9782*100 % – 0,8988*100 % = 97,82 % - 89,88 % = 7,94 %. Это

достаточно большое расхождение. А значит реальная оценка много хуже, чем мы ожидали.

3.6. Переобучение. Определение переобучения и методы борьбы с ним

«Достаточно большое расхождение» – это очень условная фраза. Но, к сожалению, нет каких-то строгих значений, которые бы говорили, что вот сейчас модель точно плохо обучилась или, наоборот, с ней все в порядке. Поэтому давайте в рамках курса примем за граничное значение 3 % или 0,03 в абсолютных значениях.

Итак, разница между оценками в 7,94 % говорит о том, что наша модель, грубо говоря, «заучила» тренировочный набор наизусть, но так и не поняла реальную закономерность, которая скрывается в данных. Отсюда следует, что она не научилась решать задачу на реальных данных, с которыми еще не сталкивалась, а значит, она непригодна для использования в реальных задачах. Данная ситуация называется переобучение модели. Для борьбы с переобучением используют разные подходы:

– увеличивают объем обучающей выборки и, что важнее, ее репрезентативность. Таким образом, должны отсеяться различные фоновые и Набор данных, использованный для проверки не очень значимые признаки. Реально большие объемы данных позволяют лучше выявить аномалии и выбросы значений;

– уменьшают сложность нейронной сети. Уменьшение количества весов нейронной сети помешает подобрать слишком сложную функцию, а значит ей поневоле придется жертвовать точностью на обучении и не подстраиваться так точно под каждую точку, как она делает на рисунках 2 и 3. Это повышает шанс подобрать хорошую обобщающую функцию;

– следят за процессом обучения на каждой эпохе и останавливают обучение, как только появились признаки переобучения.

В случае, когда мы не можем найти больше данных и уменьшение сложности модели не помогает, остается подход с остановом обучения. Тут,

конечно, можно было обойтись тестовым набором, про который мы уже говорили. Просто тестировать на нем качество модели не в конце обучения, а в конце каждой эпохи. И как только видим расхождение между оценкой на тренировочном и тестовом наборе в 3 %, останавливаем обучение.

3.7. Параметры и гиперпараметры нейронной сети

Но давайте вспомним, что процесс обучения нейронной сети способен настраивать лишь веса модели и смещения нейронов (про которые мы говорили, когда изучали функции активации). Смещения и веса нейронов – это параметры нейронной сети. Но существует ряд параметров, которые нельзя эффективно настроить в процессе обучения нейронной сети. Они задаются еще до начала обучения и в процессе обучения не изменяются (некоторые способны изменяться по окончании эпохи обучения). Такие параметры называются гиперпараметрами нейронной сети. К ним относятся:

– количество слоев нейронной сети;

– количество нейронов в слое нейронной сети;

– количество эпох обучения;

– скорость обучения;

– размер обучающей выборки;

– функции активации;

– функция ошибок.

Для наилучшего результата обучения мы должны:

– выбрать первоначальную конфигурацию нейронной сети;

– обучить ее;

– проверить результат (а для этого мы пока используем тестовый набор);

– поменять первоначальную архитектуру сети или какие-то ее гиперпараметры и повторить все снова.

И тут возникает проблема. Мы от одной конфигурации к другой показываем нейронной сети тестовые данные и, отталкиваясь от результата, меняем гиперпараметры. В конечном итоге мы приходим к качеству, которое нас

удовлетворяет. Но, к этому моменту мы переобучаемся по отношению к тестовому набору и наша оценка снова нереалистична. Отсюда следует, что нам нужен еще один набор данных.

Давайте от тренировочного набора возьмем некий процент данных. В зависимости от объема данных мы можем захватить от 10 % до 20 % (это не строгие значения, их можно произвольно менять). И будем использовать этот набор для оценки качества модели в конце каждой эпохи. Он поможет нам избежать переобучения при подборе параметров сети. Мы просто будем останавливать обучение, когда оценка модели на этом наборе будет слишком расходиться с оценкой на обучающем наборе. Его же мы используем для подбора гиперпараметров нейронной сети. Этот набор данных обычно называют проверочным (validation). Окончательно поймем, не переобучились ли мы теперь под гиперпараметры, на хорошо известном нам тестовом наборе данных. Как мы говорили ранее, тестовый набор откладывается заранее, а вот проверочный можно извлекать заново после каждой эпохи обучения следующим образом:

```
history = model.fit(x_train, y_train, batch_size=200,
        epochs=50,validation_split=0.2, verbose=1)
```

Тут уже знакомая нам функция обучения нейронной сети, но в ней мы видим новый параметр *validation_split=0.2*. Как раз он отвечает за выделения части набора на проверку. В данном случае – 20 %. Новым также является то, что мы присваиваем процесс обучения сети некоей переменной *history*. Это позволяет сохранить информацию об обучении и представить ее визуально (рис. 3.1).

```
plt.plot(history.history['accuracy'],
label='Доля правильных ответов на обучающем наборе')
plt.plot(history.history['val_accuracy'],
label='Доля правильных ответов на проверочном наборе')
plt.xlabel('Эпоха обучения')
plt.ylabel('Доля правильных ответов')
```

```
plt.legend()
plt.show()
```

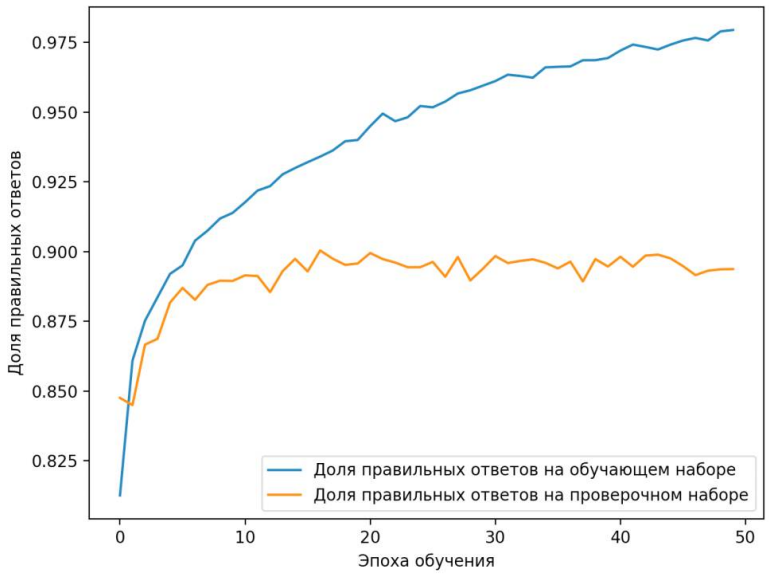


Рисунок 3.1. Кривые обучения нейронной сети

На рисунке 3.1 четко видно, что точность на проверочном наборе и на тренировочном разошлись в первых эпохах и все дальнейшее обучение было только во вред и лишь тратило ресурсы и время.

Пример решения задачи классификации на примере распознавания рукописных цифр из набора данных MNIST в файле Распознавание_рукописных_цифр_из_набора_данных_MNIST_с_помощью_Keras.ipynb и в приложении Б.

3.8. Порядок выполнения работы

1. Выбрать одну из баз данных (согласовать с преподавателем), список которых представлен на сайте http://archive.ics.uci.edu/ml/.

2. Исходные данные должны храниться в файле. Программным способом считать эти данные и сформировать обучающую, проверочную и тестовую выборки. Обучающая выборка должна включать достаточное количество примеров для обучения. Тестовая выборка может быть вполовину меньше обучающей. Примеры из тестовой выборки не должны быть включены в обучающую. Для оценки обучениямодели можно использовать проверочную

(валидационную) выборку (при достаточном количестве исходных данных) или же использовать кросс-валидацию (если данных мало).

3. Разработать программу на языке Python с использованием библиотек Keras и TensorFlow. Программа должна обеспечивать: построение и обучение НС; тестирование НС; вычисление точности и ошибки классификации; сохранение / считывание модели; вывод на экран структуру НС.

4. Согласно варианту заданий построить модель НС таким образом, чтобы точность на обучающей выборке была не 100%, но более чем 90% (желательно около 97-98%). В процессе обучения, записывайте в таблицу, какие у вас получались точности для различных моделей НС на обучающей и тестовой выборках в процессе подбора гиперпараметров.

3.9. Содержание отчета

1. Титульный лист.
2. Описание предметной области и постановка задачи.
3. Описание всех параметров НС, построение архитектуры.
4. Листинг программы.
5. Результаты работы программы: общий вид обучающей, проверочной и тестовой выборок; модель НС; таблица экспериментов подбора гиперпараметров; точность итоговой модели на обеих выборках.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Барсегян А. А. Анализ данных и процессов: учебное пособие [Электронный ресурс] / А. А. Барсегян, М. С. Куприянов, И. И. Холод, М. Д. Тесс, С. И. Елизаров. – 3-е изд., перераб. и доп. – 9Мб. — СПб.: БХВ-Петербург, 2009. – 1 файл. – Систем. требования: Acrobat Reader.

2. Дейтел Пол. Python: Искусственный интеллект, большие данные и облачные вычисления. [Электронный ресурс] / Дейтел Пол, Дейтел Харви — СПб.: Питер, 2020. — 864 с.: ил. — (Серия «Для профессионалов»). 1 файл. – Систем. требования: Acrobat Reader.

3. Антонио Джулли. Библиотека Keras – инструмент глубокого обучения. Реализация нейронных сетей с помощью библиотек Theano и TensorFlow [Электронный ресурс] Антонио Джулли, Суджит Пал / пер. с англ. Слинкин А.А. – М.: ДМК Пресс, 2018. -294 с.: ил. 1 файл. – Систем. требования: Acrobat Reader.

4. Жерон, Орельен. Прикладное машинное обучение с помощью Scikit-Learn и TensorFlow: концепции, инструменты и техники для создания интеллектуальных систем. [Электронный ресурс]. Пер. с англ. - СпБ.: ООО «Альфа-книга»: 2018. - 688 с.: ил. 1 файл. – Систем. требования: Acrobat Reader.

5. Шакла Нiwант. Машинное обучение и TensorFlow. - СПб.: Питер, 2019. - 336 с.: ил. - (Серия«Библиотека программиста»).

4. Федин Ф. О. Анализ данных. Часть 1. Подготовка данных к анализу: учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва: Московский городской педагогический университет, 2012. — 204 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/26444.html. — Режим доступа: для авторизир. пользователей

5. Федин Ф. О. Анализ данных. Часть 2. Инструменты Data Mining: учебное пособие / Ф. О. Федин, Ф. Ф. Федин. — Москва: Московский городской педагогический университет, 2012. — 308 c. — ISBN 2227-8397. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: http://www.iprbookshop.ru/26445.html. — Режим доступа: для авторизир. пользователей.

Приложение А

Листинг программы 01_Scikit_learn_DecisionTreeClassifier_iris.ipynb

Интеллектуальный анализ данных

Python Деревья решений

1. Загрузка данных

Вариант 1.

Использовать данные из библиотеки

In [ ]:
```
from sklearn.datasets import load_iris
```

In [ ]:
```
iris = load_iris()
```

In [ ]:
```
print(iris)
```

In [ ]:
```
X = iris.data
Y = iris.target
```

In [ ]:
```
print(X)
X.shape
```

In [ ]:
```
print(Y)
Y.shape
```

Вариант 2.

Подгружать свои данные, используя библиотеку Pandas как для загрузки данных так и для предварительного анализа.

2. Построение дерева

Основные параметры класса sklearn.tree.DecisionTreeClassifier:

- max_depth – максимальная глубина дерева
- max_features — максимальное число признаков, по которым ищется лучшее разбиение в дереве (это нужно потому, что при большом количестве признаков будет "дорого" искать лучшее (по критерию типа прироста информации) разбиение среди всех признаков)
- min_samples_leaf – минимальное число объектов в листе. У этого параметра есть понятная интерпретация: скажем, если он равен 5, то дерево будет порождать только те классифицирующие правила, которые верны как минимум для 5 объектов

In [ ]:
```
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

In [ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3)
X_train.shape
```

In [ ]:
```
tree_1=DecisionTreeClassifier()
```

In [ ]:
```
tree_1.fit(X_train, Y_train)
```

3. Тестирование дерева

In [ ]:
```
Y_test_predict=tree_1.predict(X_test)
print(Y_test_predict)
```

In [ ]:
```
mean_accuracy = tree_1.score(X_test, Y_test, sample_weight=None)
print(mean_accuracy)
```

4. Визуализация дерева

Вариант 1

In [ ]:
```
from sklearn import tree
```

In [ ]:
```
tree.plot_tree(tree_1)
```

Приложение Б

Листинг программы Пример_Распознавание рукописных цифр.ipynb

```
[ ] from tensorflow.keras.datasets import mnist # Библиотека с базой Mnist
    from tensorflow.keras.models import Sequential # Библиотека НС прямого распространения
    from tensorflow.keras.layers import Dense # Библиотека НС Полносвязные слои
    from tensorflow .keras import utils #Библиотека для преобразований данных
    from tensorflow .keras.preprocessing import image #Библиотека для работы с изображениями
    from google.colab import files #Библиотека для загрузки файлов
    import numpy as np #Библиотека для работы с массивами
    import matplotlib.pyplot as plt #Библиотека для отрисовки изображений
    from PIL import Image #Библиотека для работы с изображениями
    #Дает возможность отрисовывать изображения в ноутбуке
    %matplotlib inline
```

Подготовка данных для обучения сети

Загружаем набор данных с рукописными цифрами

```
[ ] # В Keras встроены средства работы с популярными наборами данных
    #Загрузка данных Mnist
    #x_train_org - входные картинки, обучающая выборка
    #y_train_org - выходные индексы, обучающая выборка
    #x_test_org - входные картинки, тестовая выборка
    #y_test_org - выходные индексы, тестовая выборка
    (x_train_org, y_train_org), (x_test_org, y_test_org) = mnist.load_data()
```

```
[ ] #  смотрим размер данных
    # обучающая выборка входные данные
    print(x_train_org.shape)
```

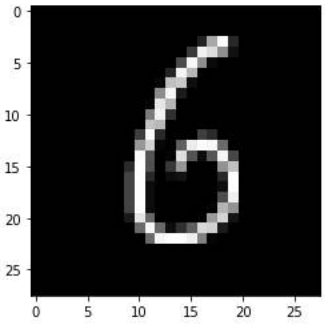
```
(60000, 28, 28)
```

```
[ ] # проверочная выборка входные данные
    print(x_test_org.shape)
```

```
(10000, 28, 28)
```

```
[ ] n = 18 #Номер картинки

    #Отрисовка картинки
    #Image.fromarray - создание картинки по массиву
    #.convert('RGBA') - преобразование в RGBA формат
    plt.imshow(Image.fromarray(x_train_org[n]).convert('RGBA'))
    plt.show()
```



Преобразование размерности данных в наборе

```
[ ] #Меняем формат входных картинок с 28x28 на 784x1
    #Это нужно для работы с полносвязным слоем (Dense)
    x_train = x_train_org.reshape(60000, 784)
    x_test = x_test_org.reshape(10000, 784)
```

Нормализация данных

```
[ ] #Нормализуем входные картинки
    #НС лучше работают, если данные нормированы от 0 до 1
    #Значения входных данных изменяется от 0 до 255
    #Если разделить на 255 данные будут нормированы от 0 до 1
    # И будем преобразовать данные в тип float
    x_train = x_train.astype('float32')
    x_train = x_train / 255
    x_test = x_test.astype('float32')
    x_test = x_test / 255
```

Работа с правильными ответами

```
[ ] print(x_train.shape)
```

```
(60000, 784)
```

Преобразуем метки в формат one hot encoding

```
[ ] # формат one hot encoding
    #В нашем случае это последовательность из 10 цифр - все 0 и одна 1 там, где правильный ответ
    #Например
    #Индекс 0 - это [1 0 0 0 0 0 0 0 0 0]
    #Индекс 2 - это [0 0 1 0 0 0 0 0 0 0]
    #Индекс 9 - это [0 0 0 0 0 0 0 0 0 1]
    #Это делается функцией utils.to_categorical(
    #Второй параметр - это количество классов, у нас - 10
    y_train = utils.to_categorical(y_train_org, 10)
    y_test = utils.to_categorical(y_test_org, 10)
```

Правильный ответ в формате one hot encoding

```
[ ] #Выводим размер y_train
    #60 тысяч примеров, каждый размера 10 (так как 10 классов)
    print(y_train.shape)
```

```
(60000, 10)
```

```
[ ] #Выводим пример одного выходного вектора
    print(y_train[n])
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
```

Создаем нейронную сеть

Создаем последовательную модель

```
[ ] #Создаём сеть прямого распространения
    #Документация  https://keras.io/api/models/sequential/
    model = Sequential()
```

```
#Добавляем в сеть полносвязные слои
#Dense - полносвязный слой
#Первый парамерт - количество нейронов, подбирается в ходе исследования
#input_dim - размер входного вектора, указывается только для первого слоя сети
#activation - активационная функция нейронов данного слоя
#документация  https://keras.io/api/layers/activations/
model.add(Dense(800, input_dim=784, activation="relu"))
model.add(Dense(400, activation="relu"))

#Выходной полносвязный слой, 10 нейронов (по количеству рукописных цифр)
#activation="softmax" - активационная функция,
model.add(Dense(10, activation="softmax"))
```

Компилируем сеть

```
#loss - функция ошибки, которую оптимизирует сеть
#categorical_crossentropy - одна из функций ошибки, подходящая для классификации нескольких классов
#optimizer - алгоритм обучения НС
#adam - один из алгоритмов обучения НС, документация  https://keras.io/api/optimizers/
#metrics - метрика качества обучения, документацияhttps://keras.io/api/metrics/
#accuracy - метрика, процент правильно распознанных примеров
model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

#Вывод структуры НС
print(model.summary())
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_3 (Dense)             (None, 800)               628000

 dense_4 (Dense)             (None, 400)               320400

 dense_5 (Dense)             (None, 10)                4010

=================================================================
Total params: 952,410
Trainable params: 952,410
Non-trainable params: 0
_________________________________________________________________
None
```

Обучаем нейронную сеть

```
[ ] #fit - функция обучения
    #x_train, y_train - обучающая выборка, входные и выходные данные
    #batch_size - размер батча, количество примеров, которое обрабатывает НС, перед изменением весов
    #epochs - количество эпох (итераций)
    #verbose - 0 - не визуализировать ход обучения, 1 - визуализировать
    #validation_split - процент данных для валидации
    model.fit(x_train, y_train, batch_size=200, epochs=20,  verbose=1, validation_split=0.2)
```

```
Epoch 1/20
240/240 [==============================] - 2s 7ms/step - loss: 0.2612 - accuracy: 0.9222 - val_loss: 0.1284 - val_accuracy: 0.9619
Epoch 2/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0887 - accuracy: 0.9731 - val_loss: 0.0855 - val_accuracy: 0.9737
Epoch 3/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0547 - accuracy: 0.9834 - val_loss: 0.0846 - val_accuracy: 0.9748
Epoch 4/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0355 - accuracy: 0.9883 - val_loss: 0.0765 - val_accuracy: 0.9774
Epoch 5/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0243 - accuracy: 0.9924 - val_loss: 0.0879 - val_accuracy: 0.9753
Epoch 6/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0174 - accuracy: 0.9945 - val_loss: 0.0884 - val_accuracy: 0.9766
Epoch 7/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0182 - accuracy: 0.9943 - val_loss: 0.0820 - val_accuracy: 0.9787
Epoch 8/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0142 - accuracy: 0.9950 - val_loss: 0.0979 - val_accuracy: 0.9756
Epoch 9/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0105 - accuracy: 0.9965 - val_loss: 0.1057 - val_accuracy: 0.9772
Epoch 10/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0109 - accuracy: 0.9961 - val_loss: 0.0956 - val_accuracy: 0.9765
Epoch 11/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0080 - accuracy: 0.9974 - val_loss: 0.1078 - val_accuracy: 0.9770
Epoch 12/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0112 - accuracy: 0.9962 - val_loss: 0.1184 - val_accuracy: 0.9724
Epoch 13/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0064 - accuracy: 0.9978 - val_loss: 0.1015 - val_accuracy: 0.9805
Epoch 14/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0054 - accuracy: 0.9982 - val_loss: 0.1003 - val_accuracy: 0.9790
Epoch 15/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0118 - accuracy: 0.9960 - val_loss: 0.1102 - val_accuracy: 0.9762
Epoch 16/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0087 - accuracy: 0.9972 - val_loss: 0.1071 - val_accuracy: 0.9793
Epoch 17/20
240/240 [==============================] - 2s 6ms/step - loss: 0.0132 - accuracy: 0.9957 - val_loss: 0.1134 - val_accuracy: 0.9778
Epoch 18/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0080 - accuracy: 0.9977 - val_loss: 0.1150 - val_accuracy: 0.9790
Epoch 19/20
240/240 [==============================] - 1s 6ms/step - loss: 0.0038 - accuracy: 0.9988 - val_loss: 0.1115 - val_accuracy: 0.9811
Epoch 20/20
240/240 [==============================] - 2s 7ms/step - loss: 0.0080 - accuracy: 0.9977 - val_loss: 0.1094 - val_accuracy: 0.9768
<keras.callbacks.History at 0x7f4bc1a745d0>
```

Сохраняем обученную нейронную сеть

Записываем обученную нейронную сеть в файл `mnist_dense.h5`

```
[ ]  #Сохраняем нейронку в файл
     model.save("mnist_NN_1.h5")
```

Проверяем, что файл сохранился

```
[ ]  #Выводим на экран список текущих файлов
     !ls
```

```
mnist_NN_1.h5  mnist_NN.h5  sample_data
```

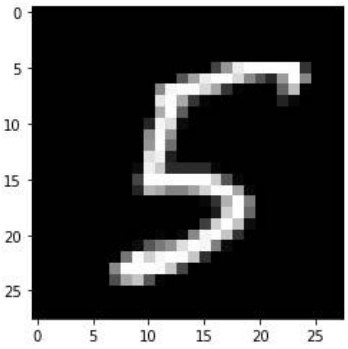
Сохраняем файлы на локальный компьютер

```
[ ]  #Скачиваем файл локально на компьютер
     files.download("mnist_NN.h5")
```

Используем сеть для распознавания рукописных цифр

```
[ ]  #Номер тестовой цифры, которую будем распознавать
     n_test = 23
```

```
[ ]  plt.imshow(Image.fromarray(x_test_org[n_test]).convert('RGBA'))
     plt.show()
```



Меняем размерность изображения и нормализуем его

```
[ ]  #Выбираем нужную картинку из тестовой выборки
     x = x_test[n_test]
```

```
[ ]  #Делаем массив из одного примеры
     x = np.expand_dims(x, axis=0)
```

```
[ ]  # смотрим размер
     x.shape
```

```
(1, 784)
```

Запускаем распознавание

```
[ ] #Распознаём ннаш пример
    prediction = model.predict(x)
```

Печатаем результаты распознавания

```
[ ] #Выводим результат, это 10 цифр
    #Сумма значений равна 1, так как финальный слой с активационной функций softmax
    print(prediction)
```

```
[[8.4793195e-23 2.6191275e-22 1.2297920e-21 1.7563566e-12 6.2885442e-23
  1.0000000e+00 3.3041314e-18 5.6347785e-23 1.0642300e-16 2.3695272e-14]]
```

Преобразуем результаты из формата one hot encoding

```
[ ] #Получаем индекс самого большого элемента
    #Это итоговая цифра, которую распознала сеть
    prediction = np.argmax(prediction)
    print(prediction)
```

```
5
```

Печатаем правильный ответ

```
[ ] #выводим правильный ответ, для сравнения
    print(y_test_org[n_test])
```

```
5
```

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ
для проведения лабораторных работ по дисциплине
«Интеллектуальный анализ данных»

Составитель:

Васяева Татьяна Александровна – кандидат технических наук, доцент кафедры автоматизированных систем управления ГОУВПО «ДОННТУ»

Ответственный за выпуск:

Секирин Александр Иванович – кандидат технических наук, доцент, заведующий кафедрой автоматизированных систем управления ГОУВПО «ДОННТУ»