Лабораторная работа № 1. Подготовка данных для построения моделей машинного обучения. Визуализация данных.

**Теоретическая часть.**

Первичная обработка данных проводится до того момента, как данные будут использоваться для обучения модели. Полученные данные очень часто имеют искажения и непригодны для обучения. При этом, в данных могут быть пропуски и выбросы. Использование таких данных при построении моделей машинного обучения приводит к неверным результатам. Подготовка данных подразумевает под собой первоначальное изучение данных, используемого для определения и планирования необходимой предварительной обработки.

Реальные данные получают из различных источников (датчики, тексты, изображения, таблицы, видео, бумажные и электронные документы). Основным моментом здесь является определение, как можно более раннем этапе проекта, какая информация критически важна. Эти данные могут содержать ошибки и повреждения, негативно влияющие на качество набора данных. Вот какими могут быть типичные проблемы с качеством данных:

* *Неполнота*: данные не содержат признаков, или в них пропущены значения.
* *Шум*: данные содержат ошибочные записи или выбросы.
* *Несогласованность*: данные содержат конфликтующие между собой записи или расхождения.

Хорошо предобработанные данные — это необходимое условие для создания качественных моделей прогнозирования. Исходя из этого, необходимо провести анализ работоспособности данных, как можно раньше обнаружить проблемы и решить, какие действия по предварительной обработке и очистке данных необходимы.

Основными задачами первичной обработки данных являются:

* *Очистка данных*: заполнение отсутствующих значений, обнаружение и удаление шумных данных и выбросов.
* *Преобразование данных*: нормализация данных для уменьшения измерений и шума.
* *Уменьшение данных*. Примеры записей или признаков данных для упрощения обработки данных.
* *Дискретизация данных*. Преобразование непрерывных признаков в категориальные признаки для простоты использования с определенными методами машинного обучения.
* *One-hot-encoding.* Данный приём используется, когда признаки являются категориальными, то есть принимают значения, между которыми нельзя установить отношения порядка.

Обработка пропущенных значений. Очистка данных

При работе с пропущенными значениями необходимо сначала определить причину их появления. Основные методы обработки пропущенных данных:

* *Удаление*. Для упрощения работы с данными зачастую прибегают к удалению записей с пропущенными значениями.
* *Фиктивная подстановка*. Замена пропущенных значений фиктивными, например, подстановка значения unknown (неизвестно) вместо категориальных или значения 0 вместо чисел.
* *Подстановка среднего значения*. Пропущенные вещественные признаки можно заменить средним значением.
* *Подстановка часто используемого элемента*. Пропущенные категориальные признаки можно заменить наиболее часто используемым элементом.
* *Подстановка по регрессии*. Использование регрессионного метода для замены пропущенных значений.

Преобразование данных

Нормализация данных позволяет масштабировать числовые значения в указанном диапазоне. Ниже представлены распространенные методы нормализации данных:

* *линейная нормализация*. Линейное преобразование данных в диапазоне, например, от 0 до 1, где минимальное и максимальное масштабируемые значения соответствуют 0 и 1 соответственно. Линейная нормализация набора произвольных данных выполняется по формуле (1).

|  |  |
| --- | --- |
|  | (1) |

где – исходный элемент, – минимальное значение признака , – максимальное значение признака *.*

* *Стандартизация*. Масштабирование данных на основе среднего значения равного 0 и стандартного отклонения равного 1 в соответствии с формулой (2).

|  |  |
| --- | --- |
|  | (2) |

где – исходный элемент, – математическое ожидание признака , – среднеквадратическое отклонение признака .

* *Десятичное масштабирование*. Масштабирование данных путем удаления десятичного разделителя значения признака.

Уменьшение данных

Существует множество различных способов, с помощью которых можно уменьшить размер данных для упрощения их обработки. В зависимости от размера данных можно применить следующие методы:

* *Выборка записей*. Создание выборки записей данных и выбор репрезентативного подмножества из общего набора данных.
* *Выборка признаков*: отбор в данных списка важнейших признаков.
* *Агрегирование*: разделение данных на группы и хранение числовых значений для каждой группы. Например, для уменьшения размера данных вы можете агрегировать числа, обозначающие ежедневный доход предприятия за 20 лет, так, чтобы указывался ежемесячный доход.

Дискретизация данных

Данные можно дискретизировать, преобразовав непрерывные значения в номинальные признаки или интервалы. Это можно сделать несколькими способами:

* *Группирование равной ширины*. Разделение диапазона всех возможных значений признака в N групп одинакового размера с последующим присвоением значений, относящихся к ячейке с соответствующим номером.
* *Группирование равной высоты*. Разделение всех возможных значений признака в N групп, содержащих одинаковое количество экземпляров, с последующим присвоением значений, относящихся к ячейке с соответствующим номером.

One-hot-encoding

Данный метод подразумевает под собой кодирование значений категориального признака бинарным вектором (т.н. дамми переменные), длина которого будет равна количеству возможных значений данного признака, и на всех его позициях будут стоять нули за исключением индекса, соответствующего текущему значению. Данная операция порой значительно увеличивает размерность задачи, но позволяет эффективно включать в модель категориальные признаки [1].

Визуализация данных

Визуализация данных — это демонстрация получения данных (информации) и помещения их в визуальную среду, например, в руководство или диаграмму. Визуализация данных особенно важна, когда дело касается больших данных и проектов анализа данных. Они используются для визуализации признаков в пространстве данных, чтобы определить основные моменты, которые могут повлиять на последующую подгонку. Визуализация требует огромного сложного объема данных для представления диаграмм или графов для быстрого усвоения информации. На основе визуализации можно получить статистический обзор более высокого уровня о том, как распределены данные, существуют ли в них выбросы и понимание как они устроены.

Для визуализации данных используется библиотека *Matplotlib* языка программирования *Python*. Она рассчитана на работу с двумерными и трёхмерными графиками.

В *Matplotlib* все объекты организованы в единую иерархию. На вершине иерархии находится конечный автомат (модель, число возможных внутренних состояний которой конечно), предоставляемый модулем *matplotlib.pyplot*. На данном уровне все функции, используемые для рисования диаграмм (линии, рисунки, текст и т.д.) применяются к текущему изображению. На уровне ниже располагается объектно-ориентированный интерфейс, используемый для создания изображений, где объектами являются само изображение, оси координат, графические примитивы и т.д.

На рисунке 1 представлены основные элементы изображения, которые можно нанести на график с помощью библиотеки *Matplotlib.* Все элементы изображения унаследованы от класса Artists (модуль *artist*) и делятся на 2 типа:

* примитивы. Стандартные графические объекты на изображении: линия (класс Line2D), прямоугольник (класс Rectangle), текст (класс Text) и т.д.;
* контейнеры. Контейнеры - объекты, внутри которых размещаются примитивы: координатная плоскость (класс Axes), оси (класс Axis) или изображение (класс Figure).

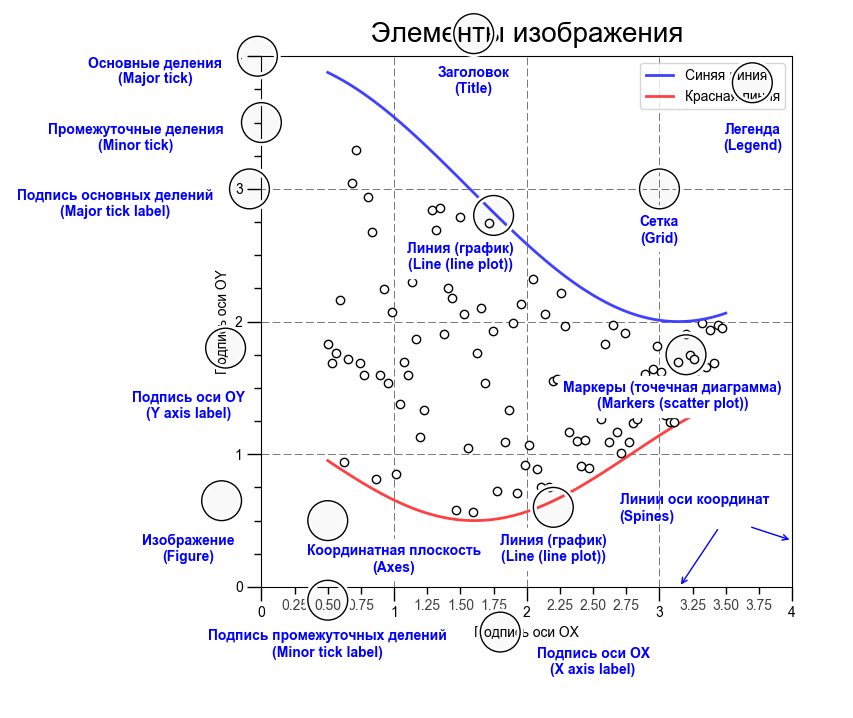


Рисунок 1 - Элементы изображения библиотеки *Matplotlib*

Кроме библиотеки *Matplotlib* существует другая библиотека *Seaborn*, созданная на ее основе. *Seaborn* такая же мощная, как и *Matplotlib*, но в то же время предоставляет большую абстракцию для упрощения графиков и привносит некоторые уникальные функции. Архитектура *Seaborn* позволяет вам быстро изучить и понять свои данные. *Seaborn* захватывает целые фреймы данных или массивы, в которых содержатся все ваши данные, и выполняет все внутренние функции, нужные для семантического маппинга и статистической агрегации для преобразования данных в информативные графики.

**Практическая часть**

Рассмотрим первичную обработку данных на примере датасета *Iris*. Этот набор данных состоит из 3 различных типов лепестков и чашелистиков ирисов (сетчатая, разноцветная и виргинская), хранящихся в формате numpy.ndarray размером 150х4. Строки являются образцами, а столбцы - длиной чашелистика, шириной чашелистика, длиной лепестка и шириной лепестка.

**Подключение библиотек**

import pandas as pd

import matplotlib as plt

import seaborn as sns

import numpy as np

from scipy import stats

**Загрузка датасета**

df = pd.read\_csv('Iris.csv')

Выведем первые пять строк набора данных с помощью метода *head* библиотеки *pandas* (рисунок 2)*.*

df.head()

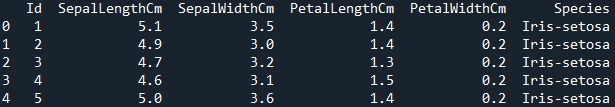


Рисунок 2 – Результат работы метода *head*

Для получения основных статистических данных по каждому признаку воспользуемся методом *describe* (рисунок 3):

df.describe()

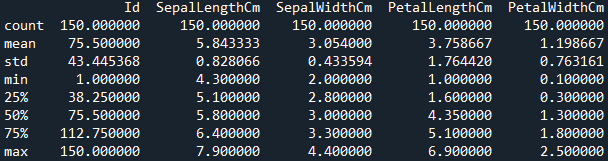


Рисунок 3 – Результат работы метода *describe*

Для извлечения отдельного столбца можно использовать конструкцию вида *DataFrame['Name']*. Например, для извлечения значений столбца *SepalLengthCm* можно воспользоваться следующей командой:

df['SepalLengthCm']

Кроме этого, можно вычислить некоторые статистические характеристики, такие как среднее, медиана и мода для данного столбца:

mean = df['SepalLengthCm'].mean()

median = df['SepalLengthCm'].median()

mode = df['SepalLengthCm'].mode()[0]

После выполнения этих методов получаем значения 5.8, 5.843333333333334 и 5.0 для среднего, медианы и моды соответственно.

Для построения гистограммы значений признака *SepalLengthCm* воспользуемся методом *displot* библиотеки seaborn (рисунок 4):

sns.displot(df['SepalLengthCm'])

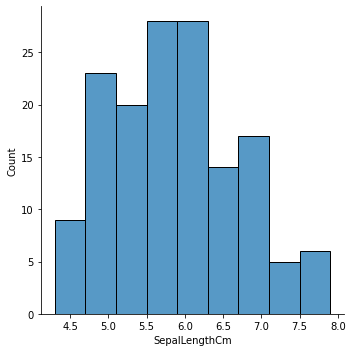


Рисунок 4 – Гистограмма значений признака *SepalLengthCm*

Для построения графика плотности признака используется функция *kdeplot* (рисунок 5):

sns.kdeplot(df['SepalLengthCm'], shade=True, legend=False)

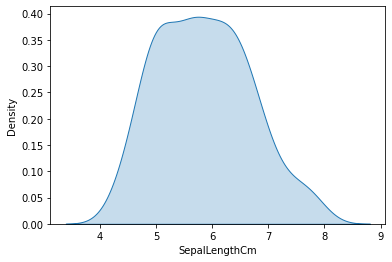


Рисунок 5 – График плотности признака *SepalLengthCm*

Очень удобной является **логическая индексация** DataFrame по одному столбцу. Выглядит она следующим образом: df[P(df['Name'])], где P — это некоторое логическое условие, проверяемое для каждого элемента столбца Name. Итогом такой индексации является DataFrame, состоящий только из строк, удовлетворяющих условию P по столбцу Name.

Например, для получения значений признака *SepalLengthCm* для класса *Iris-setosa* (рисунок 6) можно выполнить следующую строку кода:

df[df['Species'] == 'Iris-setosa']['SepalLengthCm']

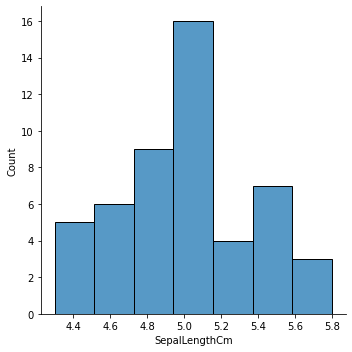


Рисунок 6 – Гистограмма распределения значений признака *SepalLengthCm* для класса *Iris-setosa*

Еще один вид графиков – *box plot* (рисунок 7):

sns.boxplot(df['SepalLengthCm'])

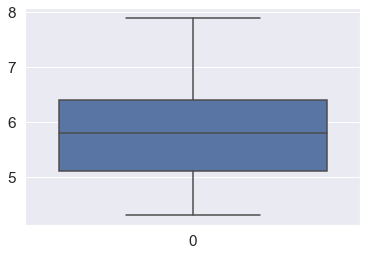


Рисунок 7 – Результат построения *box plot* для признака *SepalLengthCm*

*Box* (коробка) определяет диапазон, в который попадает 50% всех значений признака. Усы показывают минимальное и максимальное значение признака.

Для определения взаимосвязи между признаками можно использовать тип графиков *pairplot* (рисунок 8). Данный вид графиков необходим для сравнения распределения пар числовых переменных. При этом создается сетка точечных диаграмм, а также гистограмма распределения значений для каждого признака в диагональных прямоугольниках. При использовании графиков этого вида стоит обратить внимание на:

* диаграммы рассеяния, показывающие положительные линейные взаимосвязи (когда *x* увеличивается, увеличивается *y*), либо отрицательные (когда *x* увеличивается, *y* уменьшается);
* гистограммы в диагональных прямоугольниках, показывающие распределение конкретных признаков.

cols = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']

sns\_plot = sns.pairplot(df[cols])

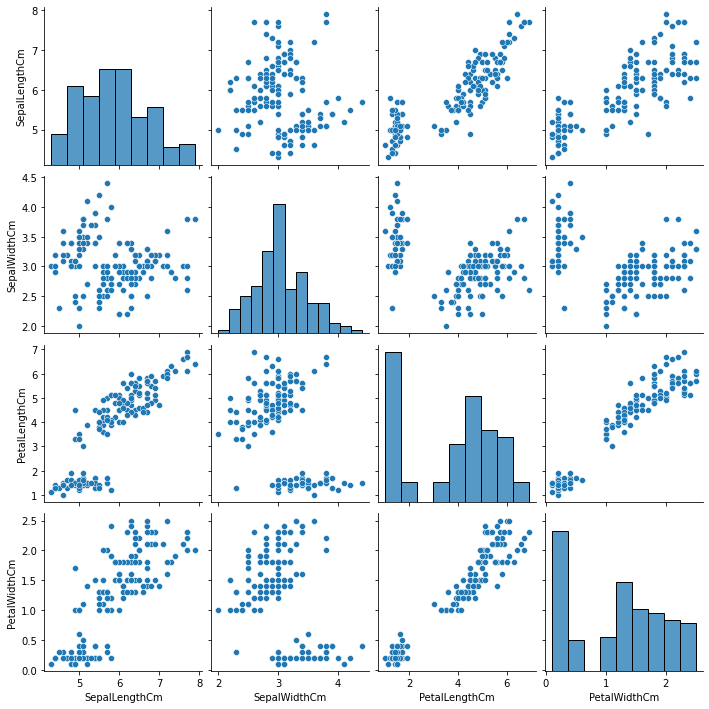


Рисунок 8 – График *pair plot*

Кроме функции *pairplot()*, для получения взаимосвязей между признаками можно также использовать другой вид графиков - *joinplot*. Функция *jointplot()* показывает совместное распределение по двум переменным (рисунок 9). Она имеет параметр *kind* который может принимать значения *scatter, reg, resid, kde, hex*.

grid = sns.jointplot(df, x ='SepalLengthCm', y = 'SepalWidthCm', kind='reg')

grid.fig.set\_figwidth(8)

grid.fig.set\_figheight(8)

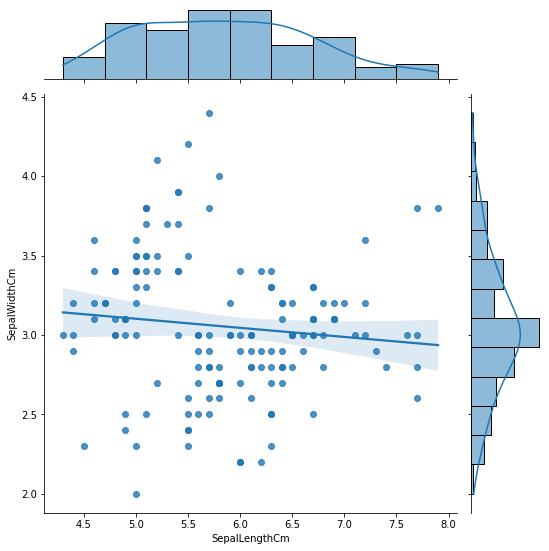


Рисунок 9 – График взаимосвязи признаков *SepalWidthCm* и *SepalLengthCm*

Для определения линейной связи между переменными используют матрицу корреляций (рисунок 10). Корреляционная матрица — это обычный инструмент, используемый для сравнения коэффициентов корреляции между различными признаками (или объектами) в наборе данных. Это позволяет понять, насколько велика (или насколько мала) корреляция между различными признаками. Это важный шаг в предварительной обработке конвейеров машинного обучения. Поскольку корреляционная матрица позволяет идентифицировать признаки с высокой степенью корреляции, они дают возможность уменьшить количество признаков, которые могут быть в представленном наборе данных.

cols = ['SepalLengthCm', 'SepalWidthCm', 'PetalLengthCm', 'PetalWidthCm']

sns.set(font\_scale=1.4)

corr\_matrix = df[cols].corr()

corr\_matrix = np.round(corr\_matrix, 2)

corr\_matrix[nabs(corr\_matrix) < 0.3] = 0

sns.heatmap(corr\_matrix, annot=True, linewidths=.5, cmap='coolwarm')

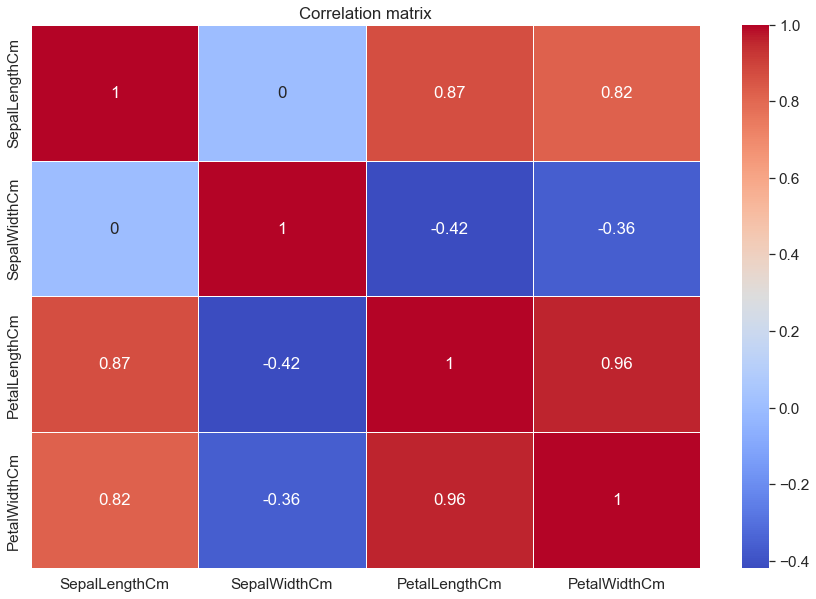


Рисунок 10 – Корреляционная матрица

**Обработка пропусков**

Проверим количество пропущенных значений для каждого из столбцов (Рисунок 11) c помощью метода из *isnull().*

missing\_num = df.isnull().sum()

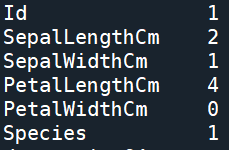


Рисунок 11 – Результат работы метода *isnull()*

При наличии в датасете пропущенных значений одним из возможных решений является удаление записей, в которых присутствуют пропущенные значения. Для удаления строк, в которых есть пропуски можно воспользоваться методом *dropna()*. Результат работы данного метода после обработки представлен на рисунке 12. Как видно из рисунка 12, после выполнения данного метода пропущенных значений в датасете не остается.

df\_correct = df.dropna(axis = 0)

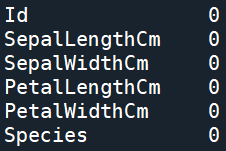


Рисунок 12 – Результат работы метода *dropna()*

Еще одним решением задачи обработки пропусков является замена пропущенных значений медианным значением, полученным из других записей. Данная операция выполняется с помощью следующих строк кода:

med = df['SepalLengthCm'].median()

df['SepalLengthCm'] = df['life\_sq'].fillna(med)

**Обработка выбросов**

Кроме, собственно, пропущенных значений во входных данных могут присутствовать выбросные значения. Для их выявления можно использовать график вида *boxplot* или *violinplot*. Например, на рисунках 13 и 14 представлены графики, с помощью которых можно увидеть, что несколько значений признака значительно превышают межквартильный размах.

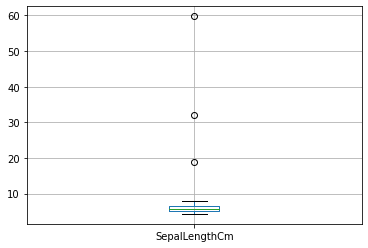


Рисунок 13 – График *boxplot* для определения выбросов

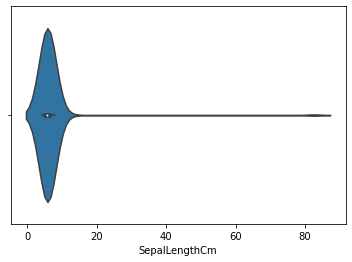


Рисунок 14 – График *violinplot* для определения выбросов

На основании среднего значения рассматриваемого признака и построенных графиков можно задать некоторый порог, при превышении которого значение будет считаться выбросом. В данном случае это может быть значение 10. Все значения признака, превышающие порог могут быть заменены средним или медианным значением. На рисунке 15 представлен график *boxplot* после обработки выбросов в представленном датасете.

df.loc[(df['SepalLengthCm'] <= 0) | (df['SepalLengthCm'] > 10), 'SepalLengthCm'] = df['SepalLengthCm'].median()

df.boxplot(column=['SepalLengthCm'])

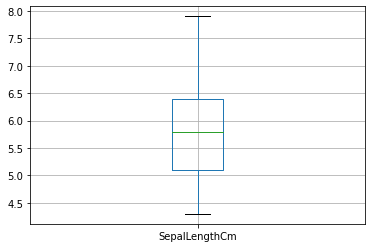


Рисунок 15 – График *boxplot после обработки выбросов*

Кроме этого, для обработки выбросов можно создавать новые признаки, в которых единицами будут отмечены строки с выбросами для данного признака. Эта процедура осуществляется следующим образом:

df['SepalLengthCm\_outlier'] = 0

df.loc[(df['SepalLengthCm'] <= 0) | (df['SepalLengthCm'] > 10), 'SepalLengthCm\_outlier'] = 1

После выполнения этих двух строк кода в датасете появится новый столбец *SepalLengthCm\_outlier,* значениями которого будут являться либо 0 – указывающий на отсутствие выброса в признаке *SepalLengthCm*, либо 1 – указывающая на наличие выброса. Таким образом можно дополнять входные данные, которые в дальнейшем могут помочь упростить обучение моделей машинного обучения.

Задание на лабораторную работу

1. Получить от преподавателя два датасета с тремя файлами (train.csv, test.csv, sample\_submission.csv) для решения задач регрессии и классификации соответственно.
2. Сделать первичную обработку данных файла train.csv (пропуски, выбросы, генерация новых признаков) для двух датасетов.
3. Произвести визуализацию данных (распределение признаков, зависимость целевой переменной от других признаков, корреляционная матрица и т.д.) для двух датасетов и сделать выводы на основе полученных графиков.
4. Сохранить обработанные датасеты в файл с расширением .csv для последующего построения моделей машинного обучения.

Описание признаков датасетов

Для задачи регрессии (цель - предсказание цены недвижимости):

* **Id**- идентификационный номер квартиры
* **DistrictId**- идентификационный номер района
* **Rooms**- количество комнат
* **Square**- площадь
* **LifeSquare**- жилая площадь
* **KitchenSquare** - площадь кухни
* **Floor** - этаж
* **HouseFloor** - количество этажей в доме
* **HouseYear** - год постройки дома
* **Ecology\_1, Ecology\_2, Ecology\_3** - экологические показатели местности
* **Social\_1, Social\_2, Social\_3** - социальные показатели местности
* **Healthcare\_1, Helthcare\_2** - показатели местности, связанные с охраной здоровья
* **Shops\_1, Shops\_2** - показатели, связанные с наличием магазинов, торговых центров
* **Price** - цена квартиры (целевая переменная)

Для задачи классификации (цель – предсказать факт невыполнения кредитных обязательств):

* **Home Ownership** - домовладение
* **Annual Income** - годовой доход
* **Years in current job** - количество лет на текущем месте работы
* **Tax Liens** - налоговые обременения
* **Number of Open Accounts** - количество открытых счетов
* **Years of Credit History** - количество лет кредитной истории
* **Maximum Open Credit** - наибольший открытый кредит (максимальная сумма, которая когда-либо была доступна клиенту)
* **Number of Credit Problems** - количество проблем с кредитом
* **Months since last delinquent** - количество месяцев с последней просрочки платежа
* **Bankruptcies** - банкротства
* **Purpose** - цель кредита
* **Term** - срок кредита
* **Current Loan Amount** - текущая сумма кредита (сумма, которую еще предстоит выплатить клиенту)
* **Current Credit Balance** - текущий кредитный баланс (сумма, которую может тратить клиент с кредитного счета)
* **Monthly Debt** - ежемесячный долг
* **Credit Score** - баллы кредитного рейтинга
* **Credit Default** - факт невыполнения кредитных обязательств (0 - погашен вовремя, 1 - просрочка)

Лабораторная работа № 2. Построение регрессионных моделей машинного обучения.

**Теоретическая часть**

Регрессионный анализ представляет собой задачу, при которой выходной результат состоит из одной или более непрерывных переменных, при этом рассматривается связь между одной зависимой и несколькими независимыми переменными. Эта связь представляется с помощью математической модели. Выбор подходящей модели основывается как на статистических доводах, так и на основе содержательного смысла моделируемой зависимости [2]. Примером решения такой задачи может служить предсказание цены летательного аппарата на основе различных характеристик (деталей, модели и т.д.).

Когда данных имеется достаточное количество, можно доверить получение выводов по этим данным в руки специальным алгоритмам, умеющим в зависимости от поставленной задачи прогнозировать интересующую нас величину, классифицировать объекты или разбивать их на группы.

На этой лабораторной работе будет рассмотрена задача обучения с учителем и построение регрессионных моделей. Основная особенность обучения с учителем состоит в том, что помимо набора данных, по которым необходимо предсказать ту или иную величину, для этих данных имеется набор правильных ответов (целевая переменная, которую необходимо предсказать). В этом случае задача заключается в том, чтобы построить модель , которая будет способна наиболее точно обобщать эти данные, то есть предсказывать правильный ответ по данным (обучающая выборка), в которых правильного ответа нет (тестовая выборка). При этом, необходимо представить функционал в виде некоторой выбранной параметрической функции с настраиваемым набором параметров . Другими словами, задача сводится к поиску неизвестных параметров на основе обучающей выборки.

Линейная регрессия

Линейные модели – это модели, которые сводятся к суммированию значений признаков с некоторыми весами согласно формуле (3). При этом, зависимость прогнозируемой переменной от других признаков будет также линейной.

|  |  |
| --- | --- |
|  | (3) |

где – настраиваемые параметры модели, – свободный коэффициент.

Оптимизация модели заключается в подборе оптимальных параметров этих весов. Учитывая, что — это скалярное произведение вектора признаков ) на вектор весов , формулу (3) можно представить в следующем виде:

|  |  |
| --- | --- |
|  | (4) |

Следует обратить внимание на тот факт, что свободный коэффициент делает модель неоднородной и затрудняет ее дальнейшую оптимизацию. Для исправления данного фактора к признаковому пространству описания объекта добавляется еще один константный признак, равный единице на каждом объекте. В этом случае вес при по смыслу будет совпадать со свободным коэффициентом. Тогда формула (4) записывается следующим образом:

|  |  |
| --- | --- |
|  | (5) |

За счет простой формы линейные модели достаточно легко обучаются и позволяют работать с зашумленными данными, небольшими выборками, контролируя при этом риск переобучения.

Для обучения модели необходимо определить критерий качества (функционал ошибки), на основе которого рассчитывают точность линейного алгоритма на обучающей или тестовой выборке. Например, в качестве критерия качества можно взять модуль отклонения целевой переменной от прогноза модели:

|  |  |
| --- | --- |
|  | (6) |

Соответствующий формуле (6) критерий качества называется *средним абсолютным отклонением* (mean absolute error, MAE). Однако, зачастую методы оптимизации в своих реализациях включают дифференцирование функций. Исходя из этого, оптимизация критерия качества (6) становится затруднительной, так как функция модуля не имеет производной в нуле. Поэтому самым частым способом измерить точность модели является квадрат разности ответа модели и значения целевой переменной. Основанный на этой функции функционал ошибки называется *среднеквадратичным отклонением* (mean squared error, MSE):

|  |  |
| --- | --- |
|  | (7) |

Дальнейшим этапом обучения модели является минимизация функционала ошибки:

|  |  |
| --- | --- |
|  | (8) |

Затем, для нахождения минимума данного функционала ошибки и весовых коэффициентов необходимо воспользоваться известным аналитическим методом (необходимое условие минимума функции) или оптимизационными методами (градиентный спуск, метод Ньютона, стохастический градиентный спуск и т.д.). На рисунке 16 приведен пример графика построения модели линейной регрессии.

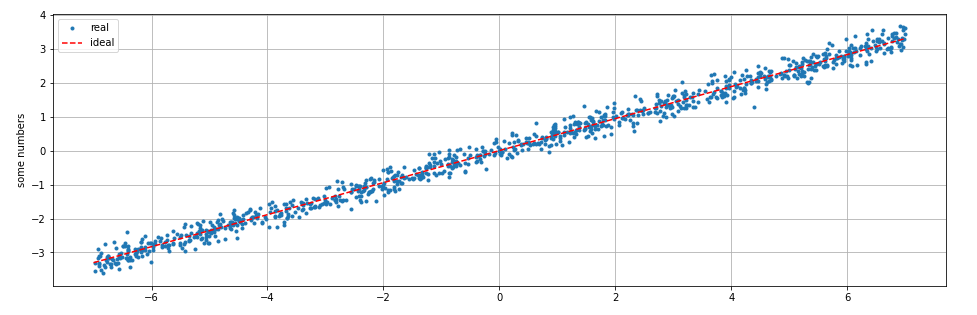


Рисунок 16 – Модель линейной регрессии

Дерево решений

Метод основан на известной структуре данных – деревьях, которые представляют собой последовательные инструкции с условиями. В листьях (терминальных узлах) деревьев стоят значения целевой функции (прогноз), а в узлах - условия перехода, определяющие, по какому из ребер идти. Если речь идет о бинарных деревьях (каждый узел производит ветвление на две части), обычно, если условие в узле истинно, то происходит переход по левому ребру, если ложно, то по правому.

Цель состоит в том, чтобы создать модель, которая предсказывает значение целевой переменной, изучая простые правила принятия решений, выведенные из характеристик данных. Дерево можно рассматривать как кусочно-постоянное приближение. Например, в приведенном ниже примере деревья решений обучаются на основе данных, чтобы аппроксимировать синусоидальную кривую с набором правил принятия решений «если-то-еще». Чем глубже дерево, тем сложнее правила принятия решений и тем лучше модель [3].

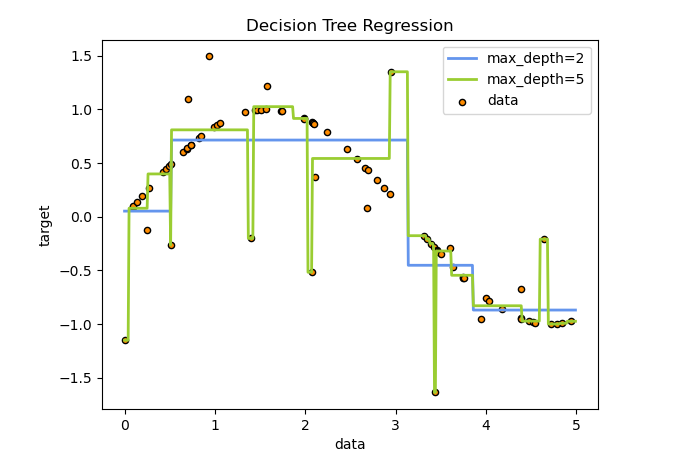


Рисунок 17 – Модель дерева решений

В основе построения дерева решений лежат «жадные» алгоритмы, допускающие локально-оптимальные решения на каждом шаге (разбиения в узлах), которые приводят к оптимальному итоговому решению. То есть при выборе одного атрибута и произведении разбиения по нему на подмножества, алгоритм не может вернуться назад и выбрать другой атрибут, даже если это даст лучшее итоговое разбиение. Следовательно, на этапе построения дерева решений нельзя точно утверждать, что удастся добиться оптимального разбиения.

В машинном обучении деревья строятся последовательно от корня к листьям (так называемый "жадный" способ). Вначале выбирается корень и критерий, по которому выборка разбивается на две. Затем то же самое делается для каждого из потомков этого корня и так далее до достаточного уровня ветвления. Задача состоит в выборе способа разбиения каждого из узлов, то есть в выборе значения порога, с которым будет сравниваться значение одного из признаков в каждом узле.

Разбиение выбирается с точки зрения некоторого заранее заданного функционала качества . Находятся наилучшие значения *i* и *j* для создания *предиката* . Параметры *i* и *t* можно выбирать перебором**:** признаков конечное число, а из всех возможных значений порога *t* можно рассматривать только те, при которых получаются различные разбиения на две подвыборки, таким образом, различных значений параметра *t* будет столько же, сколько различных значений признака в обучающей выборке. В каждой вершине производится проверка, не выполнилось ли некоторое условие останова (критерии останова рассмотрим далее), и, если оно выполнилось, разбиение прекращается, и вершина объявляется листом, и он будет содержать прогноз.

В случае регрессии в качестве ответа модели можно давать среднее (9) по выборке в листе:

|  |  |
| --- | --- |
|  | (9) |

где – множество объектов, попавших в вершину.

За функционал качества при работе с деревом решений принимается функционал вида (10):

|  |  |
| --- | --- |
|  | (10) |

где и - множества, попадающие в левое и правое поддерево, соответственно, после разбиения. – критерий информативности.

Данный функционал дает оценку качества полученной модели и принимает меньшее значение в случае, когда меньше разнообразие ответов в . Соответственно, задача обучения модели дерева решений состоит в минимизации критерия информативности и максимизации функционала .

В формуле значения критериев информативности нормируются - умножаются на долю объектов, ушедших в соответствующее подмножество. Например, если множество в узле разбилось на два подмножества размером в 980 объектов и 20 объектов, но при этом в первом подмножестве все объекты будут принадлежать к одному классу (то есть иметь минимальное значение разброса), а во втором - к разным, то в целом разбиение будет считаться хорошим, так как подавляющее большинство отсортировано правильно.

В случае регрессии разброс будет характеризоваться дисперсией, поэтому критерий информативности будет записан в виде:

|  |  |
| --- | --- |
|  | (11) |

где - среднее значение ответа в выборке X:

|  |  |
| --- | --- |
|  | (12) |

Случайный лес (Random Forest)

Случайный лес — это метаоценщик, который соответствует ряду классифицирующих деревьев принятия решений в различных подвыборках набора данных и использует усреднение для повышения точности прогнозирования и контроля над подгонкой. Случайный лес — один из самых лучших алгоритмов машинного обучения, придуманные Лео Брейманом и Адель Катлер ещё в прошлом веке. Он является одним из немногих универсальных алгоритмов. Универсальность заключается, во-первых, в том, что он хорош во многих задачах, во-вторых, в том, что есть случайные леса для решения задач классификации, регрессии, кластеризации, поиска аномалий, селекции признаков и т.д.

**RF (random forest)** — это множество решающих деревьев. В задаче регрессии их ответы усредняются. Все деревья строятся независимо по следующей схеме:

* выбирается подвыборка обучающей выборки размера samplesize (с возвращением или без) – по ней строится дерево (для каждого дерева — своя подвыборка);
* для построения каждого расщепления в дереве просматриваем max\_features случайных признаков (для каждого нового разделения — свои случайные признаки);
* выбираем наилучшие признак и разделения по нему (по заранее заданному критерию). Дерево строится, как правило, до исчерпания выборки (пока в листьях не останутся представители только одного класса), но в современных реализациях есть параметры, которые ограничивают высоту дерева, число объектов в листьях и число объектов в подвыборке, при котором проводится разделение.

В библиотеке *scikit-learn* следующая реализация RF:

*class*sklearn.ensemble.**RandomForestRegressor**(*n\_estimators=100*, *\**, *criterion='squared\_error'*, *max\_depth=None*, *min\_samples\_split=2*, *min\_samples\_leaf=1*, *min\_weight\_fraction\_leaf=0.0*, *max\_features=1.0*, *max\_leaf\_nodes=None*, *min\_impurity\_decrease=0.0*, *bootstrap=True*, *oob\_score=False*, *n\_jobs=None*, *random\_state=None*, *verboseо=0*, *warm\_start=False*, *ccp\_alpha=0.0*, *max\_samples=None*)

Опишем основные параметры.

**Число деревьев – n\_estimators**. Чем больше деревьев, тем лучше качество, но время настройки и работы RF также пропорционально увеличиваются. Обратите внимание, что часто при увеличении n\_estimators качество на обучающей выборке повышается (может даже доходить до 100%), что является признаком переобучения модели.

**Число признаков для выбора расщепления — max\_features.** График качества на тесте от значения этого параметра унимодальный, на обучении он строго возрастает. При увеличении *max\_features* увеличивается время построения леса, а деревья становятся «более однообразными». По умолчанию он равен sqrt(n) в задачах классификации и n/3 в задачах регрессии.

**Минимальное число объектов, при котором выполняется расщепление — min\_samples\_split.** Этот параметр, как правило, не очень важный и можно оставить значение по умолчанию (2). При увеличении параметра качество на обучении падает, а время построения RF сокращается.

**Ограничение на число объектов в листьях — min\_samples\_leaf.** Параметр, ограничивающий минимальное число объектов, которое может находиться в листе дерева. Чаще всего оставляют значение по умолчанию для задачи регрессии (5).

**Максимальная глубина деревьев — max\_depth.** Чем меньше глубина, тем быстрее строится и работает RF. При увеличении глубины резко возрастает качество на обучении, но и на контроле оно, как правило, увеличивается. Рекомендуется использовать максимальную глубину (кроме случаев, когда объектов слишком много и получаются очень глубокие деревья, построение которых занимает значительное время). При использовании неглубоких деревьев изменение параметров, связанных с ограничением числа объектов в листе и для деления, не приводит к значимому эффекту (листья и так получаются «большими»). Неглубокие деревья рекомендуют использовать в задачах с большим числом шумовых объектов (выбросов).

Алгоритм построения случайного леса из N деревьев выглядит следующим образом:

Для каждого :

* + Сгенерировать выборку  с помощью бутстрэпа (формировании *X* новых обучающих выборок на основе одной исходной);
  + Построить решающее дерево по выборке :

— по заданному критерию выбираем лучший признак, делаем разбиение в дереве по нему и так до исчерпания выборки;  
— дерево строится, пока в каждом листе не более  объектов или пока не достигнем определенной высоты дерева  
— при каждом разбиении сначала выбирается *m* случайных признаков из *n* исходных, и оптимальное разделение выборки ищется только среди них.

Формирование выборок на основе алгоритма бутсрэпа выглядит следующим образом. Формируются *N* новых обучающих выборок на основе одной исходной  . Для этого случайным образом выбирается k-й элемент   выборки и копируется в новую (в прежней он остается, не удаляется). Затем, эта операция повторяется *m* раз – по заданному размеру новой выборки (приблизительности 70% от количества элементов исходной выборки). Так формируется новая обучающая выборка, состоящая из элементов исходной с некоторыми повторениями, так как вполне можно несколько раз случайно отобрать один и тот же элемент (рисунок 18). После формирования одной выборки, переходят к формированию следующей и так *N* раз для *N* выборок, которые, очевидно, будут несколько отличаться друг от друга.

Итоговый ответ модели выглядит следующим образом:

|  |  |
| --- | --- |
|  | (13) |

Иными словами, ответ модели определяется средним значением по ответу каждого из построенных деревьев. В задачах регрессии рекомендуется брать значение , где – число признаков, при этом строить деревья пока в каждом листе не окажется по 5 объектов. Таким образом, случайный лес — это бэггинг (ансамблевый метаалгоритм, предназначенный для улучшения стабильности и точности алгоритмов машинного обучения) над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков.

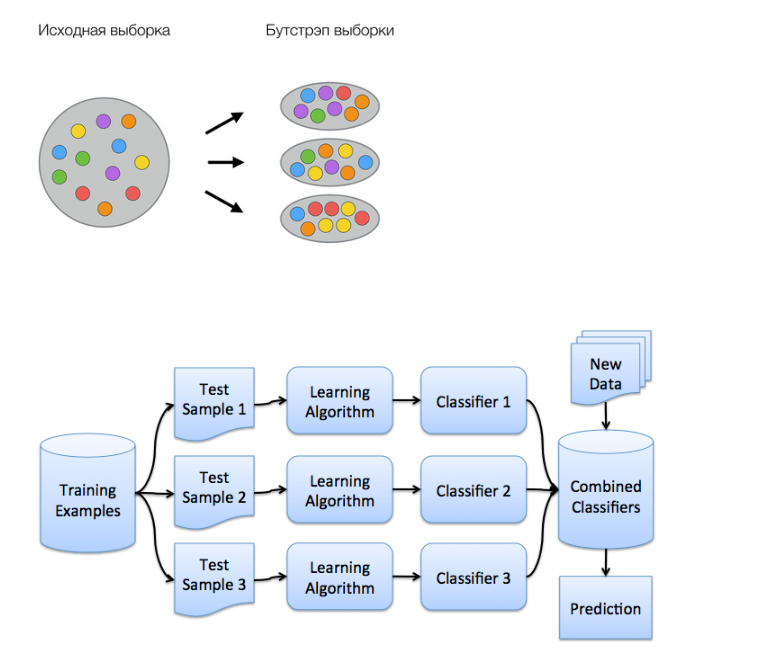


Рисунок 18 – Принцип формирования выборок на основе принципа бутсрэпа

**Практическая часть**

Задача регрессии рассматривается на примере набора данных, содержащего информацию о различных показателях студентов и полученном ими итоговым баллом. Задачей является предсказать итоговый балл студента на основе этих показателей.

Описание признаков датасета:

*Hours Studied*: общее количество часов, затраченных на учебу каждым студентом.

*Previous Scores*: баллы, полученные учащимися в ходе предыдущих тестов.

*Sleep Hours*: среднее количество часов сна, которое студент проводил в сутки.

*Sample Question Papers Practiced*: количество отработанных студентом образцов контрольных работ.

Целевая переменная:

*Performance Index* - показатель общей успеваемости каждого учащегося. Индекс успеваемости отражает академическую успеваемость студента и округлен до ближайшего целого числа. Индекс колеблется от 10 до 100.

Подключение основных библиотек и скриптов, необходимых для обучения модели.

import numpy as np

import pandas as pd

import pickle   # сохранение модели

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

Импорт моделей, линейной регрессии дерева решений, случайного леса и градиентного бустинга:

from sklearn.linear\_model import LinearRegression

from sklearn.tree import DecisionTreeRegressor, plot\_tree

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor

from sklearn.preprocessing import StandardScaler

Для обучения модели необходимо импортировать метрики оценки качества моделей.

from sklearn.metrics import mean\_squared\_error as mse, r2\_score as r2

Для визуализации работы сети используется библиотека *Image.*

from IPython.display import Image

Загрузим данные для обучения и посмотрим на первые пять строк датасета (рисунок 19):

df = pd.read\_csv(DATASET\_PATH, sep=',')

df.head()

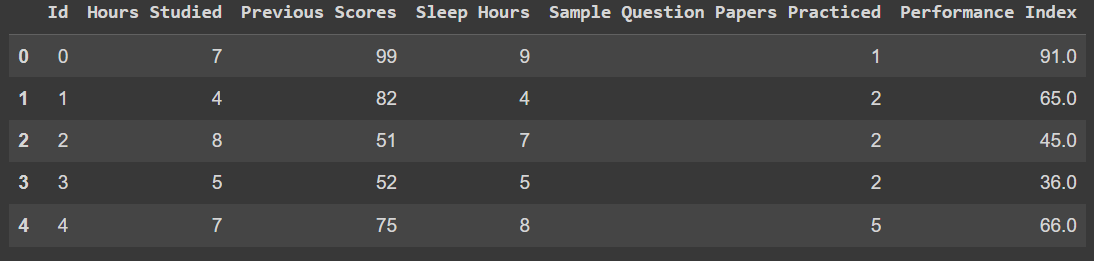


Рисунок 19 – Пример данных датасета

Выбор признаков для обучения (признаков может быть меньше после первичной обработки данных).

feature\_names = ['Hours Studied',

 'Previous Scores',

 'Sleep Hours',

 'Sample Question Papers Practiced']

target\_name = 'Performance Index'

df = df[feature\_names + [target\_name]]

df.head()

Исходя из первых пяти строк данных, можно сделать вывод о том, что данные не масштабированы. Воспользуемся одним из методов масштабирования данных – стандартизацией, для приведения к формату, когда среднее значение признаков равно 0, а стандартное отклонение – 1. Выберем все числовые признаки, создадим объект класса *StandardScaler*, и применим функцию *fit\_transform* для масштабирования данных, принимающая в качестве параметра датафрэйм с данными для масштабирования.

feature\_names\_for\_stand = df[feature\_names].select\_dtypes (include=['float32', 'float16', 'int8']).columns.tolist()

scaler = StandardScaler()

stand\_features = scaler.fit\_transform(df[feature\_names\_for\_stand])

После проведения масштабирования создают итоговый набор данных для обучения и сохраняют в формат *.csv*:

df[feature\_names\_for\_stand] = pd.DataFrame(stand\_features, columns=feature\_names\_for\_stand)

df.to\_csv(PREPARED\_DATASET\_PATH, index=False, encoding='utf-8', sep=';')

Для того чтобы работать с моделью и проверять ее качество данные необходимо разбить на тренировочную и тестовую (валидационную) выборки. Это осуществляется выполнением функции *train\_test\_split* из модуля *model\_selection* библиотеки *sklearn*. В качестве параметров данная функция принимает подготовленный после первичной обработки данных датафрейм *X*, целевую переменную *y*, размер выборки, отведенный на валидационную часть (*test\_size*), признак перемешивания выборки (*shuffle*) и признак *random\_state* для воспроизводимости данных:

X = df[feature\_names]

y = df[target\_name]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, shuffle=True, random\_state=42)

Определим метод для визуализации предсказаний модели, который строит изображение, показывающее отличие истинных и предсказанных моделью значений итогового балла. Метод принимает на вход 2 набора значений:

*true\_values –* истинные значения итогового балла

*pred\_values –* предсказанные моделью значения

def evaluate\_preds(true\_values, pred\_values, save=False):

    fig, ax = plt.subplots()

    textstr = ("R2:  " + str(round(r2(true\_values, pred\_values), 3)) + "\n" +

          "RMSE:  " + str(round(np.sqrt(mse(true\_values, pred\_values)), 3)) + "\n" +

          "MSE:  " + str(round(mse(true\_values, pred\_values), 3)))

    props = dict(boxstyle='round', facecolor='wheat', alpha=0.5)

    ax.text(0.05, 0.95, textstr, transform=ax.transAxes, fontsize=14,

            verticalalignment='top', bbox=props)

    sns.scatterplot(x=pred\_values, y=true\_values)

    ax.plot([0, 100], [0, 100], linestyle='--', color='black')  # диагональ, где true\_values = pred\_values

    plt.xlabel('Predicted values')

    plt.ylabel('True values')

    plt.title('True vs Predicted values')

    if save == True:

        plt.savefig(REPORTS\_FILE\_PATH + 'report.png')

    plt.show()

**Линейная регрессия**

Создание модели линейной регрессии осуществляется с помощью класса *LinearRegression.* Создадим объект этого класса ивоспользуемся методом *fit()* для обучения модели*.* На вход он принимает ранее созданные наборы данных:

*X\_train –* значения признаков тренировочной выборки

*y\_train –* значения целевой переменной тренировочной выборки

lr\_model = LinearRegression()

lr\_model.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающем на вход значения признаков. Выходом являются предсказания модели.

y\_train\_preds = lr\_model.predict(X\_train)

y\_train\_preds = np.clip(y\_train\_preds, a\_min=1, a\_max=100)

evaluate\_preds(y\_train, y\_train\_preds)

y\_test\_preds = lr\_model.predict(X\_test)

y\_test\_preds = np.clip(y\_test\_preds, a\_min=1, a\_max=100)

evaluate\_preds(y\_test, y\_test\_preds)

Рисунок 20 показывает различия истинных и предсказанных значений итогового балла на обучающем (а) и валидационном (б) датасетах для алгоритма линейной регрессии.

dt\_model = DecisionTreeRegressor(criterion='squared\_error',

                                 max\_depth=20,

                                 min\_samples\_leaf=30,

                                 random\_state=42)

|  |  |
| --- | --- |
| а) | б) |

Рисунок 20 – Сопоставление предсказанных и истинных значений для модели линейной регрессии на обучающем (а) и валидационном (б) датасетах

**Дерево решений**

Создание модели дерева решений осуществляется с помощью класса. *DecisionTreeRegressor.* Аналогично, создадим объект данного класс и воспользуемся методом *fit()* для обучения модели*.* Помимо исходных признаков и целевой переменной *DecisionTreeRegressor* принимает на вход следующие параметры:

*criterion* – критерий качества модели, по которому происходит разбиение в узлах дерева.

*max\_depth* – максимальная глубина дерева.

*min\_samples\_leaf* – минимальное количество наблюдений в листе дерева.

dt\_model.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающем на вход значения признаков из обучающего датасета. Выходом этого метода являются предсказания модели (итогового балла). На рисунке 21 показаны различия между истинными и предсказанными моделью значениями итогового балла на обучающем (а) и валидационном (б) датасетах для модели дерева решений.

y\_train\_preds = dt\_model.predict(X\_train)

evaluate\_preds(y\_train, y\_train\_preds)

y\_test\_preds = dt\_model.predict(X\_test)

evaluate\_preds(y\_test, y\_test\_preds)

|  |  |
| --- | --- |
| а) | б) |

Рисунок 21 – Сопоставление предсказанных и истинных значений для алгоритма дерева решений на обучающем (а) и валидационном (б) датасетах

**Случайный лес**

Создание модели случайного леса осуществляется с помощью класса *RandomForestRegressor*. Для обучения модели также создадим объект данного класса и воспользуемся методом *fit().*

rf\_model = RandomForestRegressor(criterion='squared\_error',

                                 max\_depth=20,

                                 min\_samples\_leaf=30,

                                 random\_state=42,

                                 n\_estimators=100

                                 )

*criterion* – критерий качества модели, по которому происходит разбиение в узлах дерева.

*max\_depth* – максимальная глубина дерева.

*min\_samples\_leaf* – минимальное количество наблюдений в листе дерева.

*n\_estimators* – количество деревьев.

rf\_model.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающем на вход значения признаков из обучающего датасета. На рисунке 22 показаны различия между истинными и предсказанными моделью значениями итогового балла на обучающем (а) и валидационном (б) датасетах для модели случайного леса.

y\_train\_preds = rf\_model.predict(X\_train)

evaluate\_preds(y\_train, y\_train\_preds)

y\_test\_preds = rf\_model.predict(X\_test)

evaluate\_preds(y\_test, y\_test\_preds)

|  |  |
| --- | --- |
| а) | б) |

Рисунок 22 – Сопоставление предсказанных и истинных значений для модели случайного леса на обучающем (а) и валидационном (б) датасетах

**Градиентный бустинг**

Создание модели градиентного бустинга осуществляется с помощью класса *GradientBoostingRegressor*. Для обучения модели воспользуемся уже известным методом *fit().*

gb\_model = GradientBoostingRegressor(criterion='squared\_error',

                                     max\_depth=7,

                                     min\_samples\_leaf=10,

                                     random\_state=42,

                                     n\_estimators=100)

*criterion* – критерий качества модели, по которому происходит разбиение в узлах дерева.

*max\_depth* – максимальная глубина дерева.

*min\_samples\_leaf* – минимальное количество наблюдений в листе дерева.

*n\_estimators* – количество деревьев.

gb\_model.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающем на вход значения признаков из обучающего датасета. На рисунке 23 показаны различия между истинными и предсказанными моделью значениями итогового балла на обучающем (а) и валидационном (б) датасетах для модели градиентного бустинга.

y\_train\_preds = gb\_model.predict(X\_train)

evaluate\_preds(y\_train, y\_train\_preds)

y\_test\_preds = gb\_model.predict(X\_test)

evaluate\_preds(y\_test, y\_test\_preds)

|  |  |
| --- | --- |
| а) | б) |

Рисунок 23 – Сопоставление предсказанных и истинных значений для модели градиентного бустинга на обучающем (а) и валидационном (б) датасетах

**Кросс-валидация**

Для оценки качества работы моделей машинного обучения широко применяется метод кросс-валидации. Он позволяет сравнивать между собой различные модели и выбирать наилучшую для конкретной задачи. Основным принципом кросс-валидации является разделение выборки на *K* равных частей (блоков, фолдов). При этом, обучаются *K* различных моделей, при которых один из блоков выступает в качестве валидационного, а остальные в качестве обучающих (рисунок 24). Целью описанного процесса является выбор наилучших значений гиперпараметров используемой модели машинного обучения.

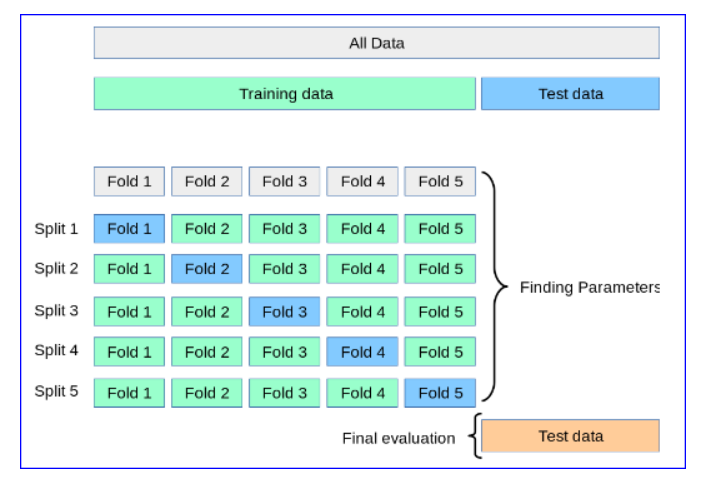


Рисунок 24 – Кросс-валидация

Рассмотрим работу метода кросс-валидации на примере модели градиентного бустинга.

gb\_model = GradientBoostingRegressor(criterion='squared\_error',

                          min\_samples\_leaf=10,

                          random\_state=42,

                         # n\_estimators=100,

                         # max\_depth=7

                         )

Класс *GridSearchCV*, использует кросс-валидацию для подбора наилучших значений гиперпараметров модели и принимает следующие параметры:

gs = GridSearchCV(gb\_model, params,

                  scoring= 'r2', # метрика

                  cv=KFold(n\_splits=5,

    random\_state=21,

                  shuffle=True),

                  n\_jobs=-1

                  )

*params –* словарь параметров и их значений для подбора.

*scoring –* метрика качества, по которой оценивается модель.

*n\_splits –* количество разбиений (блоков) в кросс – валидации.

*Shuffle* – признак перемешивания выборки.

*n*\_jobs – количество используемых ядер (-1 для использования всех процессоров).

Создаем дискретную сетку для поиска оптимального значения параметров

params = {'n\_estimators': [50, 100, 200, 400],

          'max\_depth': [3, 5, 7, 10]}

После вызова метода *fit()* происходит построение моделей со всеми возможных сочетаниями параметров модели в сетке, заданной значением *params* и оценивается качество работы модели.

gs.fit(X\_train, y\_train)

После завершения работы метода можно посмотреть наилучшие значения параметров модели с помощью метода *best\_params\_.*

gs.best\_params\_

Наилучшими значениями параметров являются:

Max\_depth = 3, n\_estimators = 200

После этого строится финальная модель с получившимися значениями параметров.

**Задание на лабораторную работу**

1. Загрузить обработанные на лабораторной работе №1 датасеты «train.csv» и «test.csv» для решения регрессионной задачи.
2. Построить различные регрессионные модели машинного обучения. Используемая метрика качества – r2.
3. Найти наилучшую модель по метрике качества (r2 должно быть больше 0,7). Построить финальную модель с наилучшими параметрами. Предсказать значения целевой переменной, используя файл «test.csv». Сформировать файл с предсказаниями по примеру файла «Sample submission».
4. Построить графики, показывающие различия между предсказанными и истинными значениями.
5. Сделать выводы по построенным моделям и графикам.

Лабораторная работа № 3. Построение классификационных моделей машинного обучения.

***Теоретическая часть***

Если выборки данных принадлежат к двум или более классам и необходимо научиться на уже размеченных данных предсказывать класс неразмеченной выборки, то в этом случае применяют классификационные модели машинного обучения. Примером задачи классификации может стать распознавание класса животных на изображении (кошка, собака, и т.д.), цель которой — присвоить каждому входному набору данных одну метку класса из конечного числа дискретных категорий. Другой способ понимания классификации — это понимание ее в качестве дискретной (как противоположность непрерывной) формы управляемого обучения, где у нас есть ограниченное количество категорий, предоставленных для N выборок; и мы пытаемся их пометить правильной категорией или классом.

Классификация объектов имеет дело с признаками этих объектов, по которым происходит процесс разделения на классы. Так как объекты могут быть представлены точками в многомерном пространстве, размерность которого определяется количеством признаков, то можно применить математическую разделимость. Некоторые не пересекающиеся множества могут быть некоторым образом разделены в пространстве, но принцип разделимости требует доказательства обоснованности применения в каждом конкретном случае [4].

Рассмотрим основные модели машинного обучения, использующиеся для решения задач классификации.

Логистическая регрессия

Логистическая регрессия – один из самых простых алгоритмов, используемых при решении задач классификации. Итак, смысл алгоритма заключается в следующем: по имеющемуся набору данных, где у каждого объекта есть метка принадлежности к классу *N*, предсказать для тестовых объектов вероятность их принадлежности к каждому из классов.

В качестве примера возьмем алгоритм логистической регрессии для задачи бинарной классификации (2 различных класса). В этом случае задача алгоритма логистической регрессии очень похожа на задачу алгоритма линейной регрессии, рассмотренного в лабораторной работе № 2: модель должна подобрать коэффициенты  (пространство исходных значений может быть разделено линейной границей (т.е. прямой) на две соответствующих классам области). В случае двух измерений — это просто прямая линия без изгибов. В случае трех — плоскость, и так далее. Эта граница задается в зависимости от имеющихся исходных данных и обучающего алгоритма (рисунок 25).

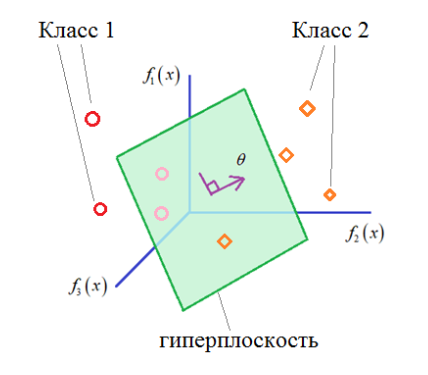


Рисунок 25 - Вид разделяющей гиперплоскости в определенном признаковом пространстве

Итак, в самом начале вычисляется следующее значение *:*

|  |  |
| --- | --- |
|  | (14) |

Далее величина помещается в сигмоидную функцию (рисунок 26):

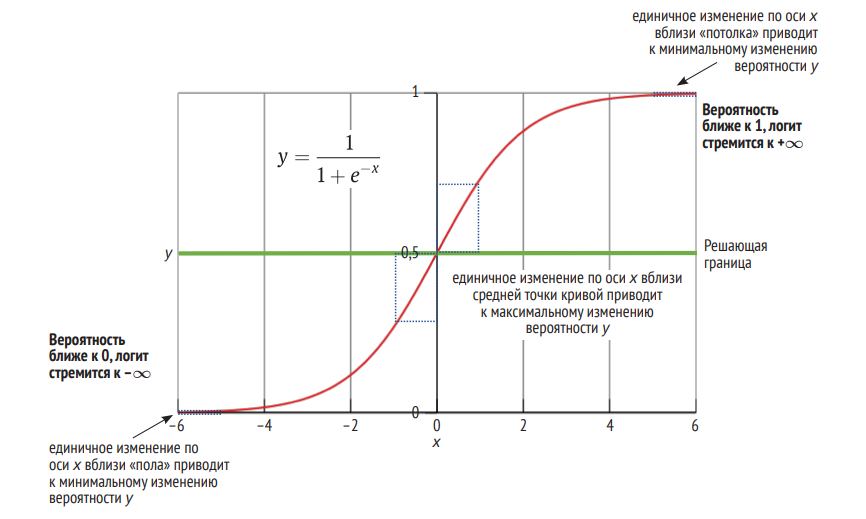


Рисунок 26 - Сигмоидальная функция

|  |  |
| --- | --- |
|  | (15) |

Эта функция используется для того, чтобы преобразовать число из промежутка в число из промежутка . При этом, значение можно интерпретировать как вероятность принадлежности объекта к классу 1. Соответственно, вероятность принадлежности к классу 0 равна .

Сразу замечаем, что независимо от того, какое значение может принимать аргумент x (неограниченная область определения функции), сигмоид ограничен двумя горизонтальными асимптотами, к которым стремится при стремлении аргумента к  (получаем ограниченный диапазон значений функции). Обычно этими асимптотами являются  и . Наличие ограниченного диапазона вещественных чисел от 0 до 1 нужно нам для прогнозирования вероятности [4].

Алгоритм логистической регрессии минимизирует следующую функцию потерь:

|  |  |
| --- | --- |
|  | (16) |

где – правильный ответ на объекте *x***,** – вычисленная вероятность принадлежности объекта *x* к классу 1.

Как и в случае линейной регрессии, для обучения алгоритма линейной классификации требуется измерять ошибку. По аналогии со средней абсолютной ошибкой и среднеквадратичной ошибкой в случае линейной классификации можно использовать естественный подход: так как возможных ответов конечное число, можно требовать полного совпадения предсказанного класса  и истинного . Тогда в качестве функционала ошибки можно использовать долю неправильных ответов:

|  |  |
| --- | --- |
|  | (17) |

Функционал ошибки (17) можно переписать в другом виде, если использовать понятие отступа *M*(сокр. от margin). Запишем формулу (14) в следующем виде:

|  |  |
| --- | --- |
|  | (18) |

Учитывая тот факт, что в задаче классификации необходимо предсказать метку класса, для получения бинарных значений берется только знак от значения .:

|  |  |
| --- | --- |
|  | (19) |

Далее, если добавить в модель один константный признак, равный единице на всем наборе данных и воспользовавшись понятием скалярного произведения перепишем выражение (19):

|  |  |
| --- | --- |
|  | (20) |

Множество точек    образует *гиперплоскость* в пространстве признаков и делит его на две части. Объекты, расположенные по разные стороны от нее, относятся к разным классам. Стоит отметить, что для некоторого объекта*x* расстояние до этой гиперплоскости будет равняться  , соответственно, при классификации важен не только знак скалярного произведения , но и его значение: чем оно выше, тем больше будет расстояние от объекта до разделяющей гиперплоскости, что будет означать, что алгоритм более уверен в отнесении объекта к данному классу. Это приводит к понятию *отступа*, который равен скалярному произведению вектора весов w на вектор признаков x, умноженному на истинное значение ответа y, которое принимает значения -1 и 1:

|  |  |
| --- | --- |
|  | (21) |

Таким образом, если скалярное произведение отрицательно, и истинный ответ равен -1, отступ будет больше нуля. Если скалярное произведение положительно, и истинный ответ равен 1, отступ также будет положителен. То есть , когда классификатор дает верный ответ, и , когда классификатор ошибается. Отступ характеризует корректность ответа, а его абсолютное значение свидетельствует о расстоянии от разделяющей гиперплоскости, то есть о мере уверенности в ответе.

Тогда, функционал ошибки (17) принимает вид:

|  |  |
| --- | --- |
|  | (22) |

Метод опорных векторов (Support Vector Machine (SVM))

Здесь будет рассмотрена задача бинарной классификации, когда различимых классов всего два и необходимо построить разделяющую линию между ними. Метод опорных векторов является одним из ведущих современных методов классификации и распознавания образов. Его также называют методом обобщенного портрета и машиной опорных векторов (Support Vector Machines, SVM) [5]. В классическом варианте алгоритма сначала, на основе прецедентов, строятся выпуклые оболочки классов, а затем строится линейная разделяющая поверхность (гиперплоскость), равноудалённая от них. В случае линейно разделимых образов построить разделяющую линию можно множеством различных способов (рисунок 27), поэтому необходимо определить правило, по которому ее необходимо строить.

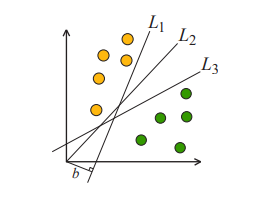


Рисунок 27 – Пример классифицирующих разделяющих прямых

Формализуем правило классификации: необходимо найти такой вектор , что для некоторого граничного значения *b* и новой точки выполняется:

|  |  |
| --- | --- |
|  | (23) |

Уравнение описывает гиперплоскость, разделяющую классы в пространстве. При этом вектор перпендикулярен искомой разделяющей прямой, а значение *b* зависит от кратчайшего расстояния между разделяющей прямой и началом координат. В задаче классификации метод SVM стремится построить между объектами разных классов "линию" (в пространствах больших размерностей это называется *гиперплоскость*) так, чтобы максимизировать расстояние от этой "линии" до объектов разных классов. Во многих случаях такой метод работает лучше, чем логистическая регрессия. Тогда модель принимает следующий вид:

|  |  |
| --- | --- |
|  | (24) |

То есть, после настройки весов алгоритма, все объекты, попадающие по одну сторону от построенной гиперплоскости, будут предсказываться как первый класс, а объекты, попадающие по другую сторону — второй класс. Чтобы разделяющая гиперплоскость как можно дальше отстояла от точек выборки (рисунок 28), ширина полосы должна быть максимальной. Если - две произвольные точки двух классов, лежащие на границе полосы, то тогда имеем:

|  |  |
| --- | --- |
|  | (25) |

Тогда, ширина полосы будет максимальной в случае минимальной нормы вектора .

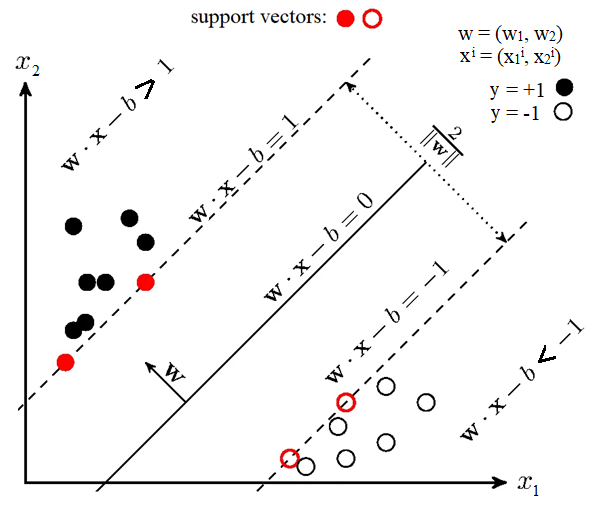


Рисунок 28 – Иллюстрация метода опорных векторов

Отступом объекта x от границы классов называется величина . Алгоритм допускает ошибку на объекте тогда и только тогда, когда отступ *M* отрицателен (когда *y* и  разных знаков). Если , то объект попадает внутрь разделяющей полосы. Если , то объект x классифицируется правильно, и находится на некотором удалении от разделяющей полосы. Т.е. алгоритм будет правильно классифицировать объекты, если выполняется условие:

|  |  |
| --- | --- |
|  | (26) |

В алгоритме метода опорных векторов используется следующий функционал качества в терминах отступа:

|  |  |
| --- | --- |
|  | (27) |

То есть, алгоритм допускает ошибку на объекте *xi*, когда значение отступа *M* отрицательно. Если заменить в функционале (27) пороговую функцию потерь кусочно-линейной верхней оценкой: , как показано на рисунке 29, то функция потерь становится чувствительной к величине ошибки. При этом, дополнительно можно ввести штраф (модель с регуляризацией), когда объект приближается к границе классов [6].

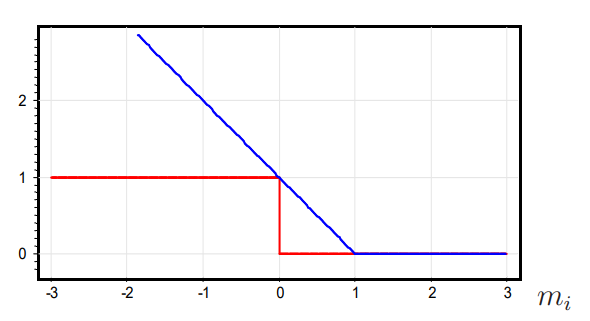


Рисунок 29 – Кусочно-линейная аппроксимация пороговой функции потерь

Тогда функционал качества принимает следующий вид:

|  |  |
| --- | --- |
|  | (28) |

Градиентный бустинг

**Бустинг** — это техника построения ансамблей, в которой предсказатели построены не независимо, а последовательно. Эта техника использует идею о том, что следующая модель будет учится на ошибках предыдущей. Они имеют неравную вероятность появления в последующих моделях, и чаще появятся те, что дают наибольшую ошибку. Предсказатели могут быть выбраны из широкого ассортимента моделей, например, деревья решений, регрессия, классификаторы и т.д. Из-за того, что предсказатели обучаются на ошибках, совершенных предыдущими, требуется меньше времени для того, чтобы добраться до реального ответа [7].

Cлучайный лес - это ансамбль деревьев небольшой глубины, строящихся независимо друг от друга. В независимости построения деревьев кроется и плюс и минус алгоритма: с одной стороны, построение деревьев можно распараллеливать и, например, организовывать на разных ядрах процессора, с другой стороны, следствием их независимости является тот факт, что для решения сложных задач требуется очень большое количество деревьев. В этих случаях (при большой выборке или большом количестве признаков) обучение случайного леса может требовать очень много ресурсов, а если для ограничения их потребления слишком ограничивать глубину деревьев, они могут не уловить все закономерности в данных и иметь большой сдвиг (и, следовательно, ошибку).

Бустинг является своеобразным решением этой проблемы: он заключается в последовательном построении ансамбля, когда деревья строятся одно за другим, и при этом каждое следующее дерево строится таким образом, чтобы исправлять ошибки уже построенного на данный момент ансамбля. При таком подходе базовые алгоритмы могут быть достаточно простыми, то есть можно использовать неглубокие деревья.

Итак, итоговый алгоритм ищется в виде взвешенной суммы базовых алгоритмов:

|  |  |
| --- | --- |
|  | (29) |

В случае построения алгоритма линейной регрессии задача состоит в минимизации среднеквадратичного функционала ошибки:

|  |  |
| --- | --- |
|  | (30) |

Учитывая тот факт, что все деревья в ансамбле, в отличие от алгоритма случайного леса, строятся последовательно, то необходимо вначале обучить первый базовый алгоритм .

|  |  |
| --- | --- |
|  | (31) |

После того как обучен базовый алгоритм, необходимо добавить в ансамбль следующий алгоритм . Однако, для обучения этого алгоритма необходимо определить невязку (32) между истинными и предсказанными базовым алгоритмом значениями.

|  |  |
| --- | --- |
|  | (32) |

Из выражения (32) следует, что если значения прибавить к предсказаниям базового алгоритма , то получится истинное значение, которое присутствует в выборке данных. Тогда, следующий алгоритм ансамбля необходимо обучить так, чтобы его предсказания были максимально близки к значению . Следовательно, функционал ошибки алгоритма запишется в виде (33).

|  |  |
| --- | --- |
|  | (33) |

Таким образом, каждый из следующих алгоритмов будет обучаться на невязках композиций предыдущих алгоритмов.

|  |  |
| --- | --- |
|  | (34) |

Таким образом, каждый новый алгоритм корректирует ошибки предыдущих, и так продолжается до момента получения приемлемой ошибки на композиции. Вектор коэффициентов  при этом называют *вектором сдвига*. Следовательно, полученный вектор сдвигадолжен минимизировать функцию потерь, то есть направлять ее в сторону наискорейшего убывания функции (антиградиента). Поэтому, при минимизации функционала ошибки

|  |  |
| --- | --- |
|  | (35) |

сдвиг на каждом шаге должен быть противоположен производной функции потерь (36) в точке .

|  |  |
| --- | --- |
|  | (36) |

Получаем, что необходимо настраивать каждый алгоритм так, чтобы он предсказывал максимально близкие ответы к вектору сдвига .  Близость ответов алгоритма к сдвигу обычно оценивается с помощью среднеквадратичной ошибки независимо от условий исходной задачи (так как исходно используемая функция потерь L уже учтена в сдвигах ).

Коэффициент λ находится по аналогии с наискорейшим градиентным спуском:

|  |  |
| --- | --- |
|  | (37) |

Обычно в качестве функции потерь в задачах регрессии принимается квадратичная функция потерь:

|  |  |
| --- | --- |
|  | (38) |

Ее производная по z принимает вид:

|  |  |
| --- | --- |
|  | (39) |

В случае задачи классификации – логистическая функция потерь:

|  |  |
| --- | --- |
|  | (40) |

И ее производная:

|  |  |
| --- | --- |
|  | (41) |

Аналогично алгоритму градиентного спуска, имеет смысл добавлять ответ каждого нового алгоритма не полностью, а с некоторым шагом , так как базовые алгоритмы обычно достаточно простые (например, деревья малой глубины), и они могут плохо приближать вектор антиградиента, и тогда вместо приближения к минимуму получаем случайное блуждание в пространстве. В градиентном бустинге такой прием называется сокращением шага.

|  |  |
| --- | --- |
|  | (42) |

Градиентный бустинг склонен к переобучению при увеличении числа итераций N или глубины входящих в него деревьев. Стоит об этом помнить при построении алгоритма и выбирать оптимальные параметры по отложенной выборке или с помощью кросс-валидации.

Алгоритм k-ближайших соседей (k-nearest neighbors)

Алгоритм k-ближайших соседей относится к классу метрических алгоритмов, то есть построенном на вычислении расстояния между объектами. Существует множество метрик, с помощью которых можно измерить расстояние. Рассмотрим основные из них:

1. Евклидово расстояние

|  |  |
| --- | --- |
|  | (43) |

1. Расстояние городских кварталов (Манхэттенское расстояние)

|  |  |
| --- | --- |
|  | (44) |

1. Метрика Минковского

|  |  |
| --- | --- |
|  | (45) |

После введения метрики, можно строить алгоритм классификации на основе критерия близости некоторого объекта к размеченным данным . Для этого удобно все объекты выборки упорядочить по возрастанию расстояния относительно *x* совместно с целевыми значениями:

|  |  |
| --- | --- |
|  | (46) |

Тогда математически классификатор в самом общем виде можно записать, следующим образом:

|  |  |
| --- | --- |
|  | (47) |

где  - вес (степень важности) i-го объекта по отношению к *x*, связанный с метрикой . Величина   неотрицательная и не возрастает при увеличении i. Фактически, здесь, мы вычисляем суммарный вес  для каждого класса и выбираем класс с наибольшей суммой.

Суть алгоритма kNN при решении задачи классификации заключается в принципе отнесения объекта к тому классу, представителей которого преобладают рядом с ним. При этом, у каждого соседа есть свой вес . То есть, теперь веса  будут принимать единичные значения для первых k ближайших векторов, а остальные оставаться нулевыми. В результате мы приходим к методу KNN со следующим алгоритмом классификации:

|  |  |
| --- | --- |
|  | (48) |

Простыми словами, из выражения (48) следует, что тестовый признак относится к тому классу, для которого число из k ближайших соседей наибольшее.

Интересной особенностью метода является то, что на этапе обучения не строится модель, а просто запоминается обучающая выборка. Вычисления начинаются именно на этапе решения задачи классификации конкретного объекта (поэтому этот алгоритм можно назвать ленивым). Часто перед работой по алгоритму kNN требуется проводить нормализацию признаков, так как они могут иметь разные единицы измерения, что может искажать расстояние между объектами.  Итогом работы с алгоритмом kNN является подбор трех параметров - количество соседей k, метрика расстояния и способ вычисления весов. Для получения лучшего качества работы алгоритма эти параметры нужно подбирать на отложенной выборке или при помощи кросс-валидации.

Практическая часть

Рассмотрим построение классификационных моделей на примере датасета *Iris*. Этот набор данных состоит из 3 различных типов лепестков и чашелистиков ирисов (сетчатая, разноцветная и виргинская), хранящихся в формате numpy.ndarray размером 150х4. Строки являются образцами, а столбцы - длиной чашелистика, шириной чашелистика, длиной лепестка и шириной лепестка.

Подключение основных библиотек и моделей необходимых для обучения моделей.

Основные библиотеки

import numpy as np

import pandas as pd

import pickle   # сохранение модели

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

Разделение датасета

from sklearn.model\_selection import train\_test\_split, KFold, GridSearchCV

Импорт моделей, логистической регрессии, метода опорных векторов и градиентного бустинга:

from sklearn import svm

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

Для оценки качества построенных моделей необходимо импортировать метрики оценки качества моделей.

from sklearn.metrics import mean\_squared\_error as mse, r2\_score as r2

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, confusion\_matrix,  accuracy\_score, classification\_report

Загрузим данные для обучения и посмотрим на первые пять строк датасета (рисунок 30):

df = pd.read\_csv(DATASET\_PATH, sep=',')

df.head()

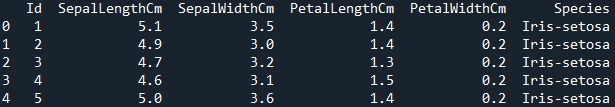


Рисунок 30 – Результат работы метода *head*

Выбор признаков для обучения

feature\_names = [ 'SepalLengthCm',

 'SepalWidthCm',

 'PetalLengthCm',

 'PetalWidthCm',]

target\_name = 'Species'

Исходя из первых пяти строк данных, можно сделать вывод о том, что данные не масштабированы. Воспользуемся одним из методов масштабирования данных – стандартизацией, для приведения к формату, когда среднее значение признаков равно 0, а стандартное отклонение – 1. Выберем все числовые признаки, создадим объект класса *StandardScaler*, и применим функцию *fit\_transform* для масштабирования данных, принимающая в качестве параметра датафрэйм с данными для масштабирования.

feature\_names\_for\_stand = df[feature\_names].select\_dtypes (include=['float32', 'float16', 'int8']).columns.tolist()

scaler = StandardScaler()

stand\_features = scaler.fit\_transform(df[feature\_names\_for\_stand])

Для многих алгоритмов классификации существенной проблемой является дисбаланс классов целевой переменной. В случаях, когда примеров одного класса существенно больше, чем примеров другого класса, возникает вероятность того, что алгоритм переучится на примерах превалирующего класса. Ситуация ещё больше усугубляется при многоклассовой классификации.

**Способы борьбы с дисбалансом классов**

1. Собрать больше данных.
2. Выбрать подходящую метрику качества.
3. Попробовать разные модели, одни модели более устойчивы к несбалансированным данным, чем другие.
4. Штраф за ошибки при прогнозе меньшего класса.
5. Undersampling и Oversampling.
6. Создание синтетических примеров для меньшего класса.

После проведения масштабирования создаем итоговый набор данных для обучения и сохраняем в формат *.csv*:

df[feature\_names\_for\_stand] = pd.DataFrame(stand\_features, columns= feature\_names\_for\_stand)

df.to\_csv(PREPARED\_DATASET\_PATH, index=False, encoding='utf-8', sep=';')

Для того чтобы работать с моделью и проверять ее качество данные необходимо разбить на тренировочную и тестовую (валидационную) выборки. Это осуществляется выполнением функции *train\_test\_split* из модуля *model\_selection* библиотеки *sklearn*. В качестве параметров данная функция принимает подготовленный после первичной обработки данных датафрейм *X*, целевую переменную *y*, размер выборки, отведенный на валидационную часть (*test\_size*), признак перемешивания выборки (*shuffle*), признак *random\_state* для воспроизводимости данных, а также признак *stratify*, позволяющий сохранить соотношение классов в тренировочной и валидационной выборках.

X = df[feature\_names]

y = df[target\_name]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33, shuffle=True, random\_state=42,  stratify=y)

Для проверки того, что соотношение классов в выборках осталось одинаковым, можно воспользоваться следующей конструкцией:

display(y\_train.value\_counts(normalize=True), y\_test.value\_counts(normalize=True))

Для оценки качества модели определим метод *get\_classification\_report*, использующий метод *classification\_report* библиотеки *sklearn.* Данный метод принимает в качестве аргументов истинные (y\_train\_true, y\_test\_true) и предсказанные моделью (y\_train\_pred, y\_test\_pred) метки классов. Метод *get\_classification\_report* вычисляет значения основных классификационных метрик: *accuracy*, *precision* (взвешенный (*weighted*) и усредненный по классам (*macro*)), *recall* (взвешенный (*weighted*) и усредненный по классам (*macro*)) и *f1-score* (взвешенный (*weighted*) и усредненный по классам (*macro*)). Далее осуществляется выбор метрики оценки качества модели. В зависимости от решаемой задачи необходимо выбрать нужную метрику. Чаще всего выбирают *f1-score*, так как эта метрика в своем расчете учитывает *precision* и *recall* и дает их среднюю оценку.

def get\_classification\_report(y\_train\_true, y\_train\_pred, y\_test\_true, y\_test\_pred):

    print('TRAIN\n\n' + classification\_report(y\_train\_true, y\_train\_pred))

    print('TEST\n\n' + classification\_report(y\_test\_true, y\_test\_pred))

    print('CONFUSION MATRIX\n')

    print(pd.crosstab(y\_test\_true, y\_test\_pred))

**Логистическая регрессия**

Создание модели логистической регрессии осуществляется с помощью класса *LogisticRegression.* Создадим объект этого класса ивоспользуемся методом *fit()* для обучения модели*.* На вход он принимает ранее созданные наборы данных:

*X\_train –* значения признаков тренировочной выборки

*y\_train –* значения целевой переменной тренировочной выборки

logreg = LogisticRegression()

logreg.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающий на вход значения признаков из обучающего и тестового датасетов. Выходом данного метода являются предсказания классов объектов.

pred\_train = logreg.predict(X\_train)

pred\_test = logreg.predict(X\_test)

Для оценки точности модели воспользуемся методом *get\_classification\_report:*

get\_classification\_report(y\_train, pred\_train, y\_test, pred\_test)

*y\_train –* значения целевой переменной тренировочной выборки

*y\_test –* значения целевой переменной тестовой выборки

*pred\_train* – значения выхода модели для тренировочной выборки (метка класса)

*pred\_test* – значения выхода модели для тестовой выборки (метка класса)

На рисунке 31 представлен результат работы метода *get\_classification\_report.*

Из рисунка 31 следует, что значение *f1-score (macro)* на обучающей выборке равно 0,98, а на тестовой – 0,91. Полученный результат считают высоким и на данном этапе можно применять полученную модель. Однако, разница между полученными значениями говорит о небольшом переобучении модели, поэтому применим и другие алгоритмы машинного обучения для решения классификационной задачи и получим значения метрики *f1-score (macro)* для сравнения с уже построенными моделями.

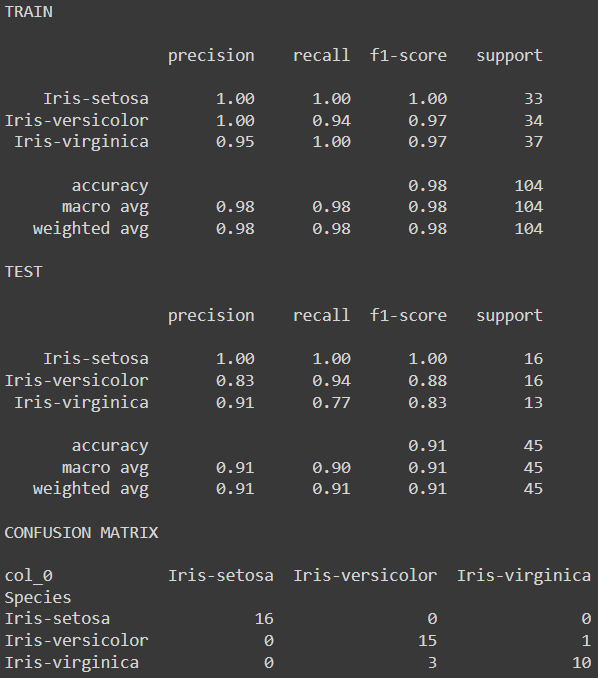


Рисунок 31 – Результат работы метода *get\_classification\_report*

**Метод опорных векторов**

Создание модели метода опорных векторов осуществляется с помощью класса *SVC.* Создадим объект этого класса ивоспользуемся методом *fit()* для обучения модели*.*

vectmach = svm.SVC()

vectmach.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающий на вход значения признаков из обучающего и тестового датасетов. Выходом данного метода являются предсказания классов объектов.

pred\_train = vectmach.predict(X\_train)

pred\_test = vectmach.predict(X\_test)

Для оценки точности модели воспользуемся методом *get\_classification\_report:*

get\_classification\_report(y\_train, pred\_train, y\_test, pred\_test)

На рисунке 32 представлен результат работы метода *get\_classification\_report.*

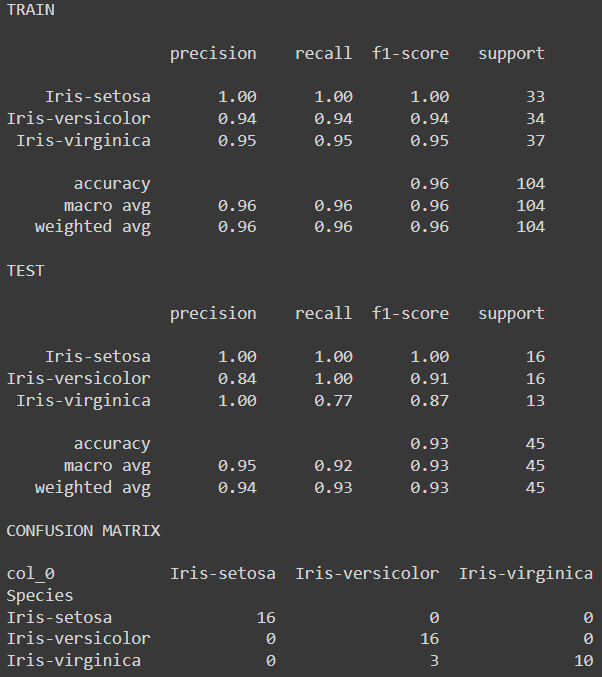


Рисунок 32 – Результат работы метода *get\_classification\_report*

Из рисунка 32 следует, что значение *f1-score (macro)* на обучающей выборке равно 0,96, а на тестовой – 0,93. Данный результат модели метода опорных векторов является лучшим, по сравнению с моделью логистической регрессии, поскольку значение метрики на тестовой выборке выросло.

**Градиентный бустинг**

Создание модели градиентного бустинга осуществляется с помощью класса *GradientBoostingClassifier.* Создадим объект этого класса ивоспользуемся методом *fit()* для обучения модели*.*

GB = GradientBoostingClassifier(n\_estimators=100, learning\_rate=1,

                                 max\_depth=1, random\_state=0)

*max\_depth* – максимальная глубина дерева.

*min\_samples\_leaf* – минимальное количество наблюдений в листе дерева.

*n\_estimators* – количество деревьев.

*learning\_rate –* скорость обучения

GB.fit(X\_train, y\_train)

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающий на вход значения признаков из обучающего и тестового датасетов. Выходом данного метода являются предсказания классов объектов.

pred\_train = GB.predict(X\_train)

pred\_test = GB.predict(X\_test)

Для оценки точности модели воспользуемся методом *get\_classification\_report:*

get\_classification\_report(y\_train, pred\_train, y\_test, pred\_test)

На рисунке 33 представлен результат работы метода *get\_classification\_report.*

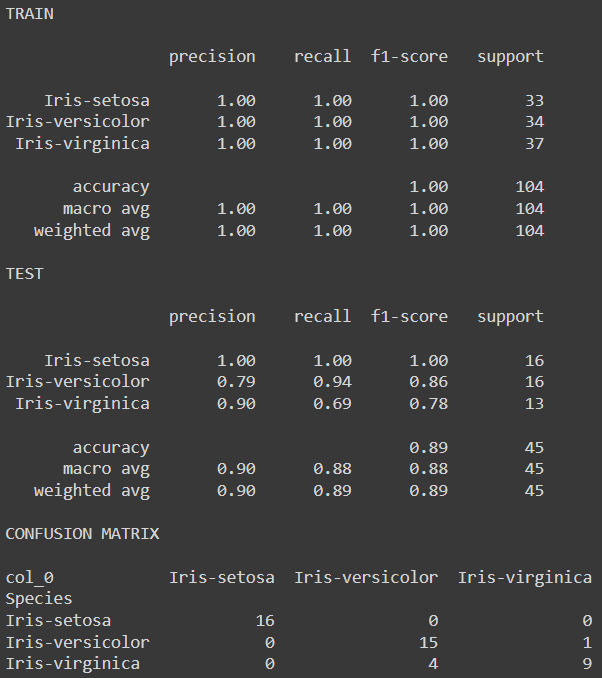


Рисунок 33 – Результат работы метода *get\_classification\_report*

Из рисунка 33 следует, что значение *f1-score (macro)* на обучающей выборке равно 1, а на тестовой – 0,88. Данный результат модели градиентного бустинга стал хуже двух предыдущих построенных моделей. При этом, в данной модели присутствует переобучение. Следовательно, модель градиентного бустинга не следует рассматривать в качестве финальной модели при решении поставленной задачи.

**K-ближайших соседей**

Создание модели k-ближайших соседей осуществляется с помощью класса *KNeighborsClassfier.* Создадим объект этого класса ивоспользуемся методом *fit()* для обучения модели*.*

nbrs = KNeighborsClassifier(n\_neighbors=2)

nbrs.fit(X\_train, y\_train)

*n\_neighbors* – количество соседей

Для получения прогноза итогового балла воспользуемся методом *predict()*, принимающий на вход значения признаков из обучающего и тестового датасетов. Выходом данного метода являются предсказания классов объектов.

pred\_train = nbrs.predict(X\_train)

pred\_test = nbrs.predict(X\_test)

Для оценки точности модели воспользуемся методом *get\_classification\_report:*

get\_classification\_report(y\_train, pred\_train, y\_test, pred\_test)

На рисунке 34 представлен результат работы метода *get\_classification\_report.*

Из рисунка 34 следует, что значение *f1-score (macro)* на обучающей выборке равно 1, а на тестовой – 0,85. Данный результат модели k-ближайших соседей хуже трех предыдущих построенных моделей. Кроме этого, отмечаем факт переобучения построенной модели. Следовательно, модель k-ближайших соседей не следует рассматривать в качестве финальной модели при решении поставленной задачи.

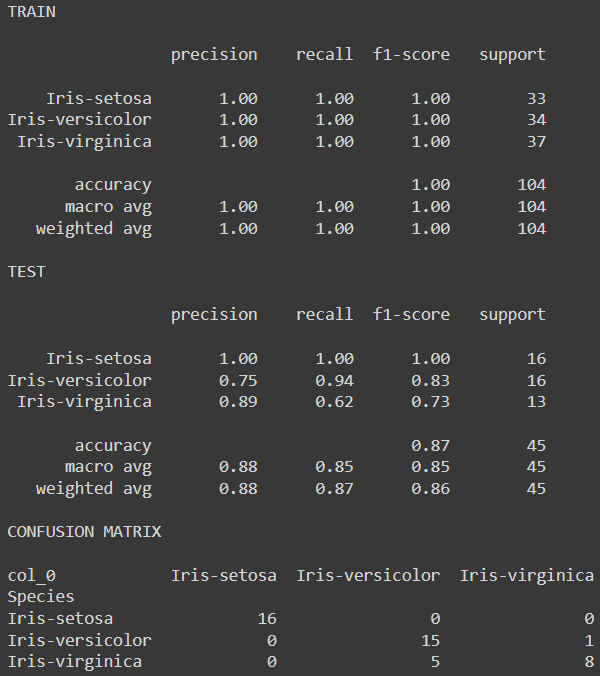
**

Рисунок 34 – Результат работы метода *get\_classification\_report*

**Задание на лабораторную работу**

1. Загрузить обработанные на лабораторной работе №1 датасеты «train.csv» и «test.csv» для решения задачи классификации.
2. Построить различные классификационные модели машинного обучения. Используемая метрика качества – *f-score* (при параметре ).
3. Найти наилучшую модель по метрике качества (*f-score* должно быть больше 0,5). Построить финальную модель с наилучшими параметрами. Предсказать значения целевой переменной, используя файл «test.csv». Сформировать файл с предсказаниями по примеру файла «Sample submission».
4. Вывести на экран значения метрик (accuracy, precision, recall, f-score) для каждого алгоритма. Сравнить полученные результаты
5. Сделать выводы по построенным моделям и получившимся значениям метрик.

Лабораторная работа № 4. Применение алгоритмов кластеризации.

Теоретическая часть

В предыдущих лабораторных работах были рассмотрены алгоритмы обучения с учителем, в которых изначально есть размеченная выборка данных с известными ответами на них и тестовая выборка, на которой проверялось качество обучения. В данной лабораторной работе будут рассмотрены алгоритмы обучения без учителя – когда в роли обучающей выборки выступает просто набор объектов и он же выступает в качестве тестовой выборки. Таким образом, задача состоит в сопоставлении объектов и проставлении меток так, чтобы объекты с одной и той же меткой были похожи. То есть все объекты в пространстве признаков нужно разделить на группы, найти структуру в данных. Такой процесс называется кластеризацией. По-простому задачу кластеризации можно сформулировать так: имеется множество точек, которые скапливаются в сгустки, нужно найти возможность относить точки к тому или иному сгустку и предсказывать, в какой сгусток попадет новая точка. Логично возникает вопрос, как измерять качество кластеризации. Есть большое количество инструментов оценки качества кластеризации, они разделяются на *внутренние* (основанные только на свойствах выборки и кластеров) и *внешние* (использующие данные об истинном распределении объектов по кластерам, если оно известно).

Примерами внутренних метрик могут быть:

* Внутрикластерное расстояние (также называется компактностью кластеров, cluster cohesion):

|  |  |
| --- | --- |
|  | (49) |

где *K* - количество кластеров,  - центр кластера. Этот функционал нужно минимизировать, так как в идеальном случае все объекты в одном кластере одинаковы, и расстояние между ними равно нулю.

* Межкластерное расстояние (отделимость кластеров, cluster separation):

|  |  |
| --- | --- |
|  | (50) |

Функционал (50), наоборот, нужно максимизировать, так как объекты из разных кластеров должны максимально различаться, то есть иметь максимальное расстояние между собой.

Среднее внутрикластерное расстояние (среднее расстояние внутри каждого кластера, просуммированное по всем кластерам) и среднее межкластерное расстояние (минимизируется и максимизируется, соответственно, по аналогии с двумя первыми функционалами):

|  |  |
| --- | --- |
|  | (51) |

где  - количество элементов в кластере под номером k.

* Индекс Данна (Dunn Index):

|  |  |
| --- | --- |
|  | (52) |

где  - расстояние между кластерами *k* и *k′* (между их центрами),  - внутрикластерное расстояние для кластера *k*. Этот функционал требуется максимизировать.

Внешние метрики используются, если есть дополнительные знания о кластеризуемой выборке, например, известно истинное распределение по кластерам. Задачу можно рассматривать как задачу многоклассовой классификации с использованием соответствующих метрик. В этом случае примерами могут быть:

* Rand Index:

|  |  |
| --- | --- |
|  | (53) |

Входящие в формулу (53) обозначения встречались при изучении классификационных алгоритмов и матриц ошибок. Здесь это количество пар объектов , которые принадлежат одному кластеру и одному классу (TP), одному кластеру, но разным классам (TN), разным кластерам, но одному классу (FP), разным кластерам и разным классам (FN). Этот индекс оценивает, сколько пар объектов, находившихся в одном классе, и пар объектов, находившихся в разных классах, сохранили это состояние после работы алгоритма. Он принимает значение от 0 до 1, где 1 - полное совпадение полученных кластеров и исходными классами, 0 - полное отсутствие совпадений.

* Jaccard Index:

|  |  |
| --- | --- |
|  | (54) |

Индекс (54) похож на предыдущий, но он не учитывает пары объектов, находящихся в разных кластерах и разных классах. Имеет такую же область определения, как и Rand Index.

* F-мера:

|  |  |
| --- | --- |
|  | (55) |

В данном случае *precision* и *recall* определяются как:

|  |  |
| --- | --- |
|  | (56) |

где  - количество объектов , принадлежащих кластеру  и классу ; = - размер кластера ; = - размер класса .

Далее рассмотрим основные алгоритмы кластеризации.

Алгоритм k-средних (k-means)

Одним из самых часто используемых алгоритмов кластеризации является алгоритм k-means. Для его реализации необходимо сделать следующие шаги:

* выбрать количество кластеров *k*, на которые будут делиться данные;
* случайным образом выбрать в пространстве данных *k* точек (центроидов) - центров будущих кластеров;
* для каждой точки из выборки посчитать, к какому из центроидов она ближе;
* переместить каждый центроид в центр выборки, отнесенной к этому центроиду, определив его как среднее арифметическое всех точек кластера:

|  |  |
| --- | --- |
|  | (56) |

* повторить шаги 3-4 до сходимости алгоритма (обычно это оценивается по величине смещения центроида после каждого шага - сходимость означает непревышение смещения какого-то заданного значения).

Проблемой метода является необходимость знать число кластеров, на которые будет делиться выборка. В случае, когда это число неизвестно, вариантом ее решения может быть последовательная кластеризация на разное число кластеров (например, от 1 до 10) с последующим анализом качества работы алгоритма, например, по сумме квадратов внутрикластерных расстояний (57)

|  |  |
| --- | --- |
|  | (57) |

выбирается такое число кластеров, начиная с которого при увеличении количества кластеров функционал падает незначительно.

Еще одним способом определить оптимальное количество кластеров на которые необходимо разделить выборку является так называемый «метод локтя» [8]. Суть этого метода применительно к задаче кластеризации заключается в рассмотрении суммарного межкластерного расстояния как целевой функции, зависящей от числа кластеров. Оптимальное число кластеров при этом определяется как число, при достижении которого падение суммарного межкластерного расстояния (рисунок 35) становится незначительным. Недостаток метода заключается в том, что «локоть» далеко не всегда явно выражен и не может быть однозначно идентифицирован.

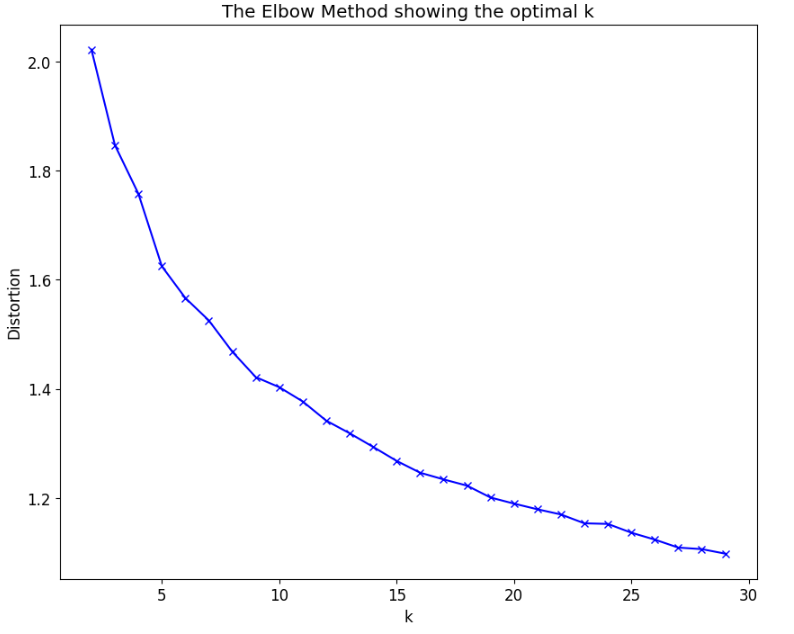


Рисунок 35 - Визуализация метода локтя

На основе рисунка 35 можно сделать вывод, что оптимальное количество кластеров, на которое нужно разбить выборку равняется приблизительно 7-8.

Кластеризация DBSCAN

Алгоритм **DBSCAN** (**Density-based spatial clustering of applications with noise**) развивает идею кластеризации с помощью выделения связных компонент [9]. Прежде чем перейти к построению графа, введём понятие плотности объектов выборки в пространстве признаков. Плотность в DBSCAN определяется в окрестности каждого объекта выборки  как количество других точек выборки в шаре . Кроме радиуса  окрестности в качестве гиперпараметра алгоритма задается порог  по количеству точек в окрестности. Визуализация метода представлена на рисунке 36.

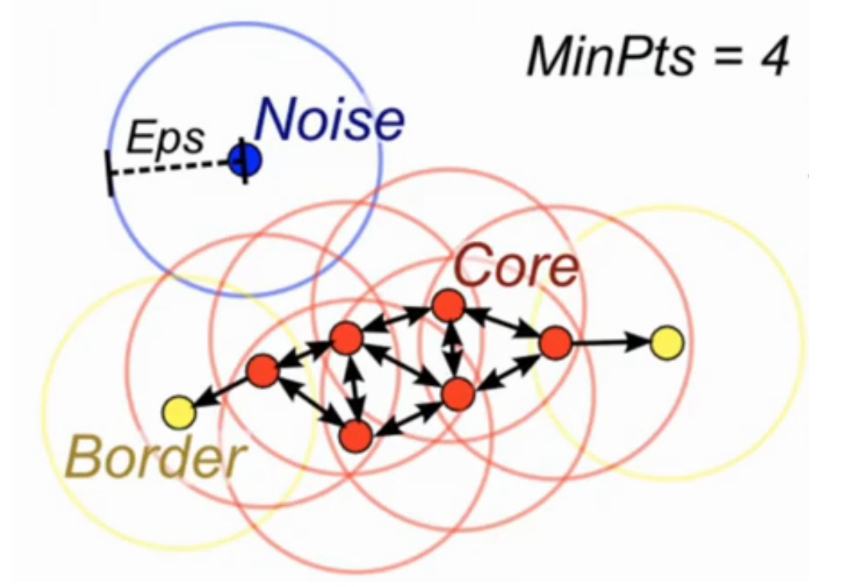


Рисунок 36 - Визуализация метода кластеризации DBSCAN

Далее все получаемые объекта разделяются на три типа:

* Внутренние точки (core points). Это такие объекты в  окрестности которых попало не менее объектов.
* Граничные точки (border points) - точки в  окрестности которых попали внутренние точки, но их количество меньше заданного .
* Шумовые точки (noise points) – точки в  окрестности которых не попало ни одной внутренней точки или содержатся только граничные точки.

Алгоритм кластеризации выглядит следующим образом:

1. Задаются параметры  – радиус шара (окружности) и – минимальное количество точек, которые попали в границы шара (окружности).
2. Основные точки, у которых есть общая окрестность объединяются в единый кластер.
3. Каждая граничная точка относится к тому кластеру, в который попала ближайшая к ней основная точка.
4. Шумовые точки убираются из рассмотрения и не приписываются ни к какому кластеру.

Удобство DBSCAN заключается в том, что он сам определяет количество кластеров, а также в том, что метод успешно справляется даже с достаточно сложными формами кластеров. Кластеры могут иметь вид протяжённых лент или быть вложенными друг в друга как концентрические гиперсферы. На изображении ниже показан пример выделения кластеров достаточно сложной формы с помощью DBSCAN (рисунок 37):

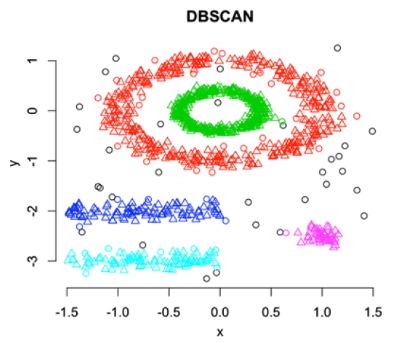


Рисунок 37 - Результат работы метода DBSCAN

Иерархическая агломеративная кластеризация

Агломеративные алгоритмы относятся к иерархичеcким методам кластеризации. Иерархические алгоритмы кластеризации строят иерархию разбиений выборки *S* от одного до *|S|* кластеров, поэтому для работы алгоритма не нужно давать на вход число кластеров. Кроме того, они не требуют какие-либо данные о структуре итоговых кластеров. Вся эта информация может использоваться уже после завершения процесса кластеризации [10].

Сам агломеративный алгоритм кластеризации можно описать следующим образом. Сначала каждый элемент из множества *S* помещается в отдельный кластер. Затем начинается итерационный процесс слияния кластеров: на каждом шаге выбираются два ближайших кластера и соединяются в один; когда все элементы оказываются в одном кластере, алгоритм заканчивается. Способ определения ближайших кластеров зависит от выбранного метода.

Возможны различные способы определения результатов кластеризации, но в настоящее время как правило используется так называемая пошаговая дендрограмма, представленная на рисунке 38.

**Методы объединения кластеров:**

* Single linkage — минимум попарных расстояний между точками () из двух кластеров
* Complete linkage — максимум попарных расстояний между точками () из двух кластеров
* Average linkage — среднее попарных расстояний между точками () из двух кластеров
* Centroid linkage — расстояние между центроидами двух кластеров

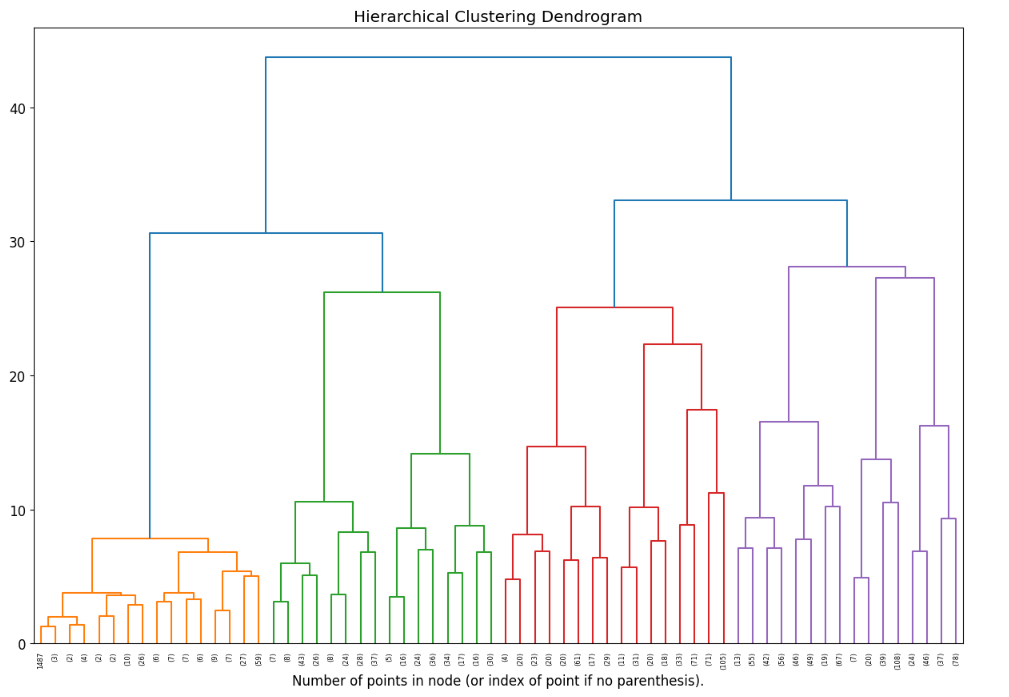


Рисунок 38 – Дендрограмма объединения кластеров в агломеративной кластеризации

Вертикальные линии на дендрограмме показывают расстояние между кластерами. Поэтому, чем меньше по длине эта линия, тем больше похожи друг на друга данные этих кластеров, так как их центры находятся рядом друг с другом. Таким образом, получаем наглядную визуализацию древовидной структуры процесса кластеризации. В частности, на дендрограмме можно визуально заметить, в какой момент происходит скачок расстояний между кластерами, и попытаться определить «естественное» количество кластеров в нашей задаче. Для этого нужно провести горизонтальную линию на дендрограмме таким образом, чтобы расстояние между кластерами было большим. При этом, количество пересечений с вертикальными линиями и будет являться ответом на вопрос об оптимальном количестве кластеров.

Сам алгоритм агломеративной кластеризации выглядит следующим образом:

1. Считаем, что каждая точка – кластер.
2. Сортируем попарные расстояния между центрами кластеров по возрастанию (исходя из выбранного метода объединения точек).
3. Пару ближайших кластеров склеиваем в один и пересчитываем центр кластера.
4. Повторяем п. 2 и 3 до тех пор, пока все данные не склеятся в один кластер.

Практическая часть

Рассмотрим алгоритмы кластеризации на примере датасета *Iris*. Этот набор данных состоит из 3 различных типов лепестков и чашелистиков ирисов (сетчатая, разноцветная и виргинская), хранящихся в формате numpy.ndarray размером 150х4. Строки являются образцами, а столбцы - длиной чашелистика, шириной чашелистика, длиной лепестка и шириной лепестка.

Подключение основных библиотек и моделей необходимых для обучения моделей.

Основные библиотеки

import numpy as np

import pandas as pd

import pickle   # сохранение модели

Для вычисления расстояния между точками признаков импортируем класс *cdist.*

from scipy.spatial.distance import cdist

Для стандартизации данных в работе будут использованы классы *StandardScaler, RobustScaler*.

from sklearn.preprocessing import StandardScaler, RobustScaler

Далее импортируем классы всех рассматриваемых алгоритмов кластеризации.

from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering

from sklearn.neighbors import KNeighborsClassifier

Для понижения размерности данных обучающей выборки будем использовать классы *PCA* и *TNSE*.

from sklearn.decomposition import PCA

from sklearn.manifold import TSNE

Подключение библиотек, необходимых для визуализации данных

import matplotlib

import matplotlib.pyplot as plt

import seaborn as sns

Определим метод *display\_clusters\_distribution* для визуализации распределения классов по кластерам. Метод принимает на вход два набора значений:

*unique\_labels* – массив с номерами кластеров

*labels\_counts* – массив с количеством элементов кластеров

def display\_clusters\_distribution(unique\_labels, labels\_counts):

    plt.figure(figsize=(8,5))

    plt.bar(unique, counts)

    plt.xlabel('Clusters')

    plt.xticks(unique)

    plt.ylabel('Count')

    plt.title('Clusters distribution')

    plt.show()

Для визуализации разбиения на кластеры объявляются функции *display\_components\_in\_2D\_space* и *display\_components\_in\_3D\_space.*

*components\_df* – структура данных содержащая информацию о координатах точек в пространстве признаков.

*labels* – структура данных, содержащая информацию о принадлежности каждой точки к определенному кластеру.

def display\_components\_in\_2D\_space(components\_df, labels=None):

    components\_with\_labels\_df = pd.concat([components\_df, pd.DataFrame(labels)], axis=1)

    if labels is not None:

        components\_with\_labels\_df.plot(kind='scatter', x='component\_1', y='component\_2',

                                         c=components\_with\_labels\_df.iloc[:, -1], cmap=plt.get\_cmap('jet'),

                                         alpha=0.5, figsize=(15,10))

    else:

        components\_with\_labels\_df.plot(kind='scatter', x='component\_1', y='component\_2', alpha=0.5, figsize=(15,10))

    plt.xlabel('component\_1')

    plt.ylabel('component\_2')

    plt.title('2D mapping of objects')

    plt.show()

Далее определим метод *reduce\_dims*, выполняющий понижение размерности данных исходного набора данных. Входными данными являются:

*df –* исходная обучающая выборка.

*dims –* размерность выходных данных после понижения.

*method –* желаемый метод понижения размерности.

*perplexity –* число ближайших соседей для метода понижения размерности TSNE.

def reduce\_dims(df, dims=2, method='pca', perplexity=30):

    assert method in ['pca', 'tsne'], 'Неверно указан метод'

    if method=='pca':

        dim\_reducer = PCA(n\_components=dims, random\_state=42)

        components = dim\_reducer.fit\_transform(df)

    elif method == 'tsne':

        dim\_reducer = TSNE(n\_components=dims, learning\_rate=250, random\_state=42, perplexity=perplexity)

        components = dim\_reducer.fit\_transform(df)

    else:

        print('Error')

    colnames = ['component\_' + str(i) for i in range(1, dims+1)]

    return dim\_reducer, pd.DataFrame(data = components, columns = colnames)

Загрузим данные обучающую выборку и посмотрим на первые пять строк датасета (рисунок 39):

df = pd.read\_csv(DATASET\_PATH, sep=',')

df.head()

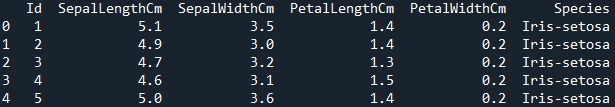


Рисунок 39 – Результат работы метода *head*

Выбор признаков для кластеризации

feature\_names = [ 'SepalLengthCm',

 'SepalWidthCm',

 'PetalLengthCm',

 'PetalWidthCm',]

X = df[feature\_names]

Перед кластеризацией необходимо провести стандартизацию признаков при помощи методов класса *RobustScaler* (или других известных классов для масштабирования данных*:*

scaler = RobustScaler()

colnames = X.columns

X\_scaled = pd.DataFrame(scaler.fit\_transform(X), columns=colnames)

Также, проведем процедуру уменьшения размерности данных при помощи ранее описанной функции *reduce\_dims.*

dim\_reducer2d, components\_2d = reduce\_dims(X\_scaled, dims=2, method='pca')

components\_2d.head(5)

Выведем первые 5 строк получившегося набора данных (рисунок 40).

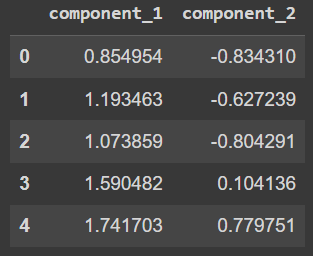


Рисунок 40 – Результат работы метода *head*

Далее рассмотрим работу каждого из описанных методов кластеризации.

Для определения оптимального количества кластеров, на которые необходимо разбить выборку применим метод локтя.

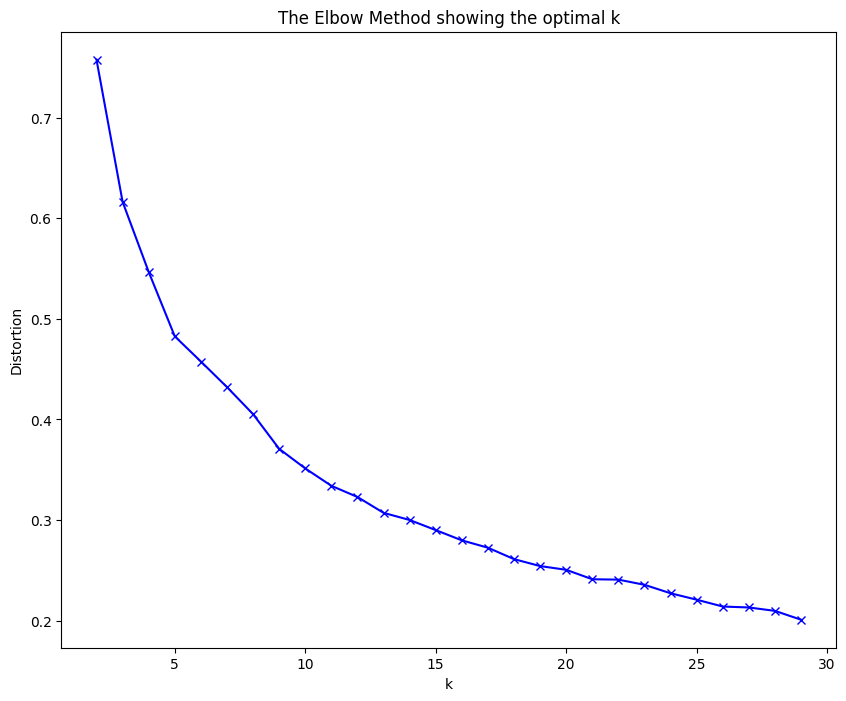


Рисунок 41 – Применение метода локтя для набора данных

Исходя из рисунка 41, можно сделать вывод о том, что оптимальное количество кластеров равно 3-4.

Алгоритм k-средних

Создание модели k-средних осуществляется с помощью класса *KMeans.*

kmeans\_3 = KMeans(n\_clusters=3, random\_state=42)

*n\_clusters* – желаемое количество кластеров

Вычисление центров всех кластеров и предсказание метки кластера для всех точек пространства осуществляется с помощью функции *fit\_predict.*

labels\_clast\_3 = kmeans\_3.fit\_predict(X\_train\_scaled)

labels\_clast\_3 = pd.Series(labels\_clast\_3, name='clusters\_3')

Далее построим гиистограмму распределения (рисунок 42) элементов выборки по кластерам.

unique, counts = np.unique(labels\_clast\_3, return\_counts=True)

display\_clusters\_distribution(unique, counts)

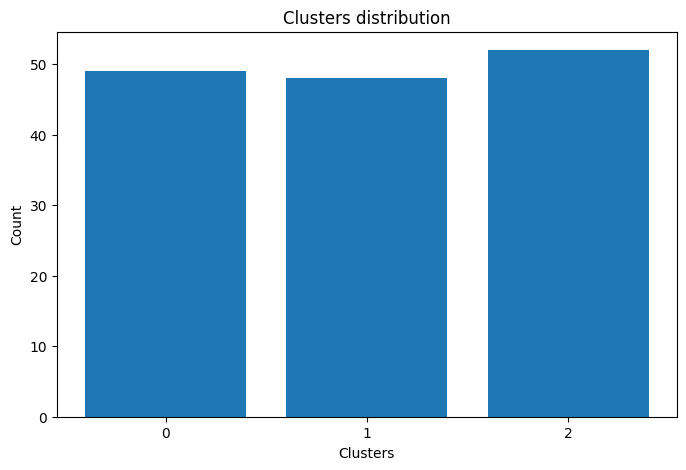


Рисунок 42 – Распределение выборки по кластерам для модели k-means

Построим график разбиения выборки на кластеры.

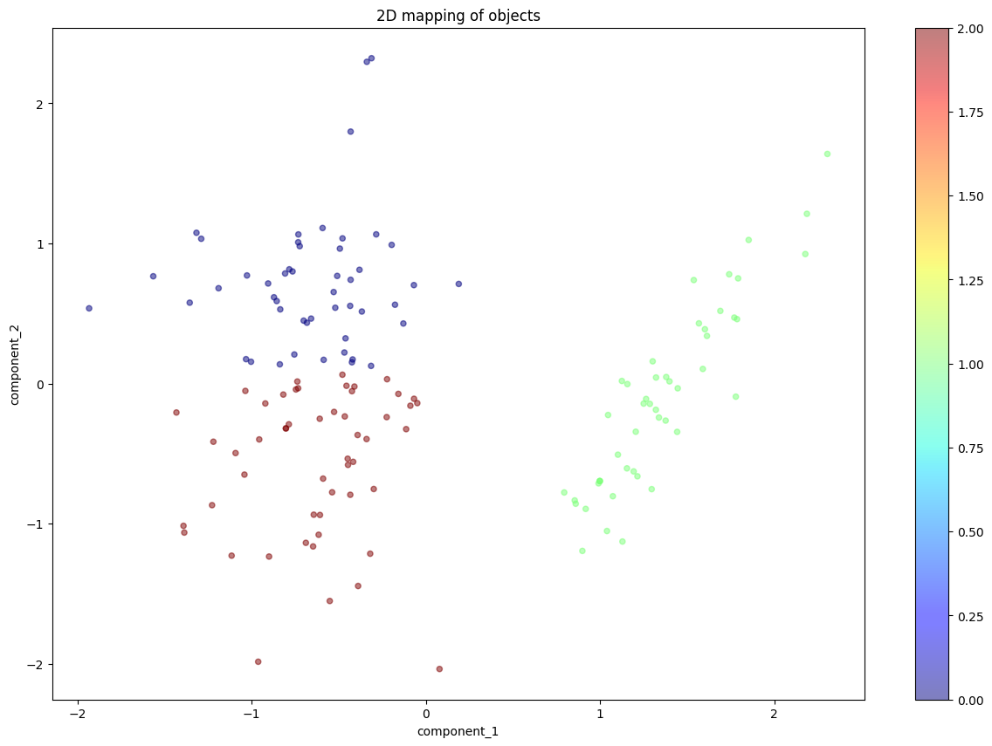


Рисунок 43 – График разбиения на кластеры для модели k – средних

Алгоритм DBSCAN

Создание модели DBSCAN осуществляется с помощью класса *DBSCAN.*

Dbscan = DBSCAN(eps=0.5, min\_samples=7)

*eps –* радиус окрестности

*min\_samples –* минимальное количество точек, которые попали в границы окрестности

Вычисление центров всех кластеров и предсказание метки кластера для всех точек пространства осуществляется с помощью функции *fit\_predict.*

Labels\_clast = dbscan.fit\_predict(X\_scaled)

labels\_clast = pd.Series(labels\_clast, name=’clusters\_dbscan’)

Далее построим гистограмму распределения (рисунок 44) элементов выборки по кластерам.

Unique, counts = np.unique(labels\_clast\_3, return\_counts=True)

display\_clusters\_distribution(unique, counts)

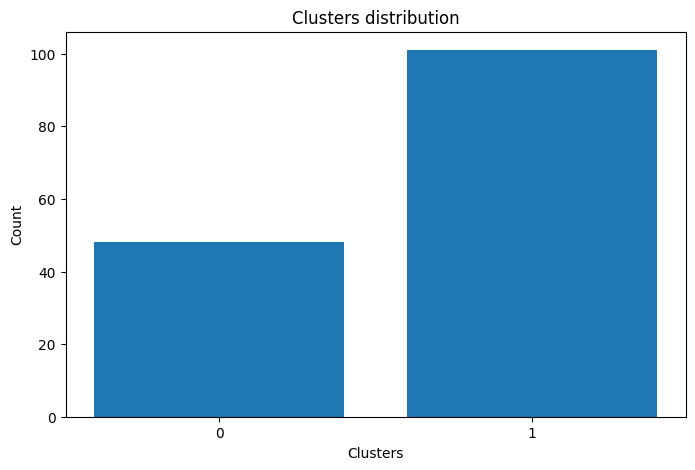


Рисунок 44 – Распределение выборки по кластерам для модели DBSCAN

Построим график разбиения выборки на кластеры для модели DBScan (рисунок 45).

Display\_components\_in\_2D\_space(components\_2d, labels\_clast)

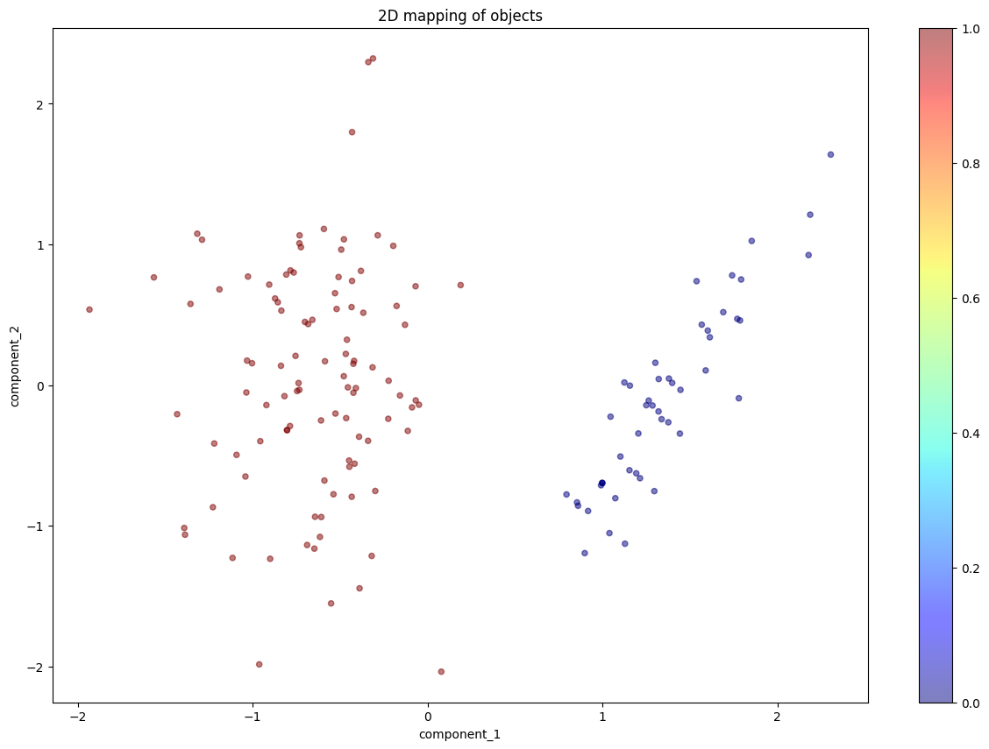


Рисунок 45 – График разбиения на кластеры для модели DBSCAN

Иерархическая агломеративная кластеризация

Создание модели агломеративной кластеризации осуществляется с помощью класса *AgglomerativeClustering.*

Aggl = AgglomerativeClustering(n\_clusters=3)

*n\_clusters –* количество кластеров

Вычисление центров всех кластеров и предсказание метки кластера для всех точек пространства осуществляется с помощью функции *fit\_predict.*

Labels\_clast = aggl.fit\_predict(X\_train\_scaed)

labels\_clast = pd.Series(labels\_clast, name=’clusters\_aggl’)

Далее построим гистограмму распределения (рисунок 46) элементов выборки по кластерам.

Unique, counts = np.unique(labels\_clast, return\_counts=True)

display\_clusters\_distribution(unique, counts)

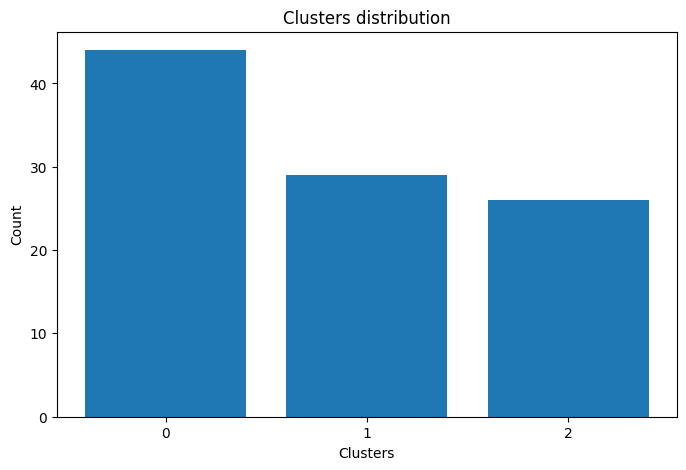


Рисунок 46 – Распределение выборки по кластерам для модели агломеративной кластеризации

display\_components\_in\_2D\_space(components\_2d, labels\_clast)

Построим график разбиения выборки на кластеры для модели агломеративной кластеризации (рисунок 47).

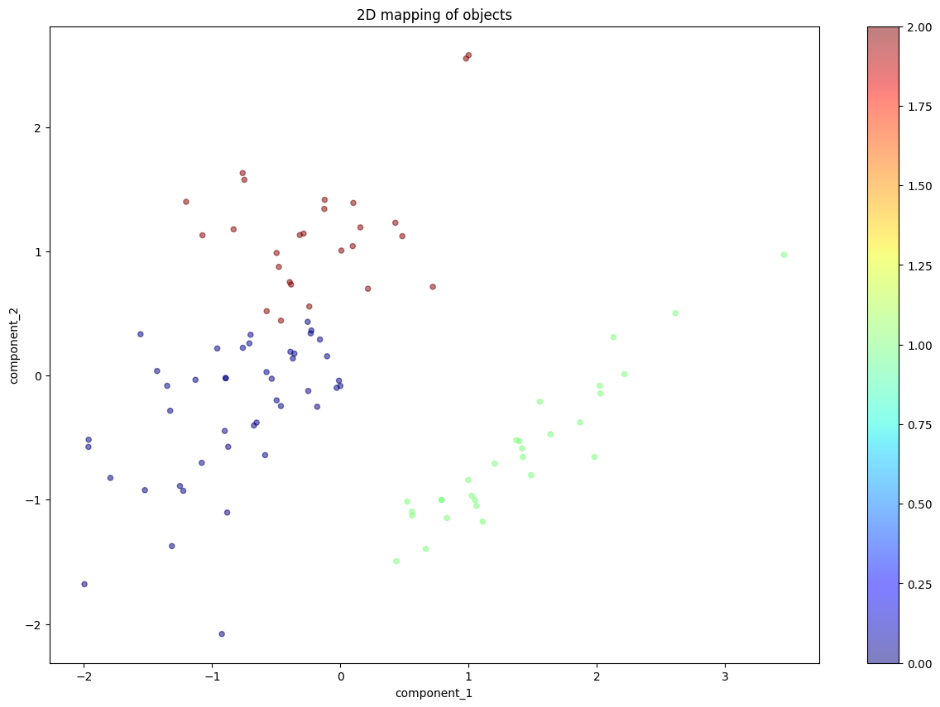


Рисунок 47 – График разбиения на кластеры для модели агломеративной кластеризации

Построим дендограмму объединения классов:

plt.figure(figsize=(15, 10))

plt.title('Hierarchical Clustering Dendrogram')

# plot the top three levels of the dendrogram

plot\_dendrogram(model, truncate\_mode='level', p=3)

plt.xlabel("Number of points in node (or index of point if no parenthesis).")

plt.show()

Результат построения дендрограммы показан на рисунке 48.

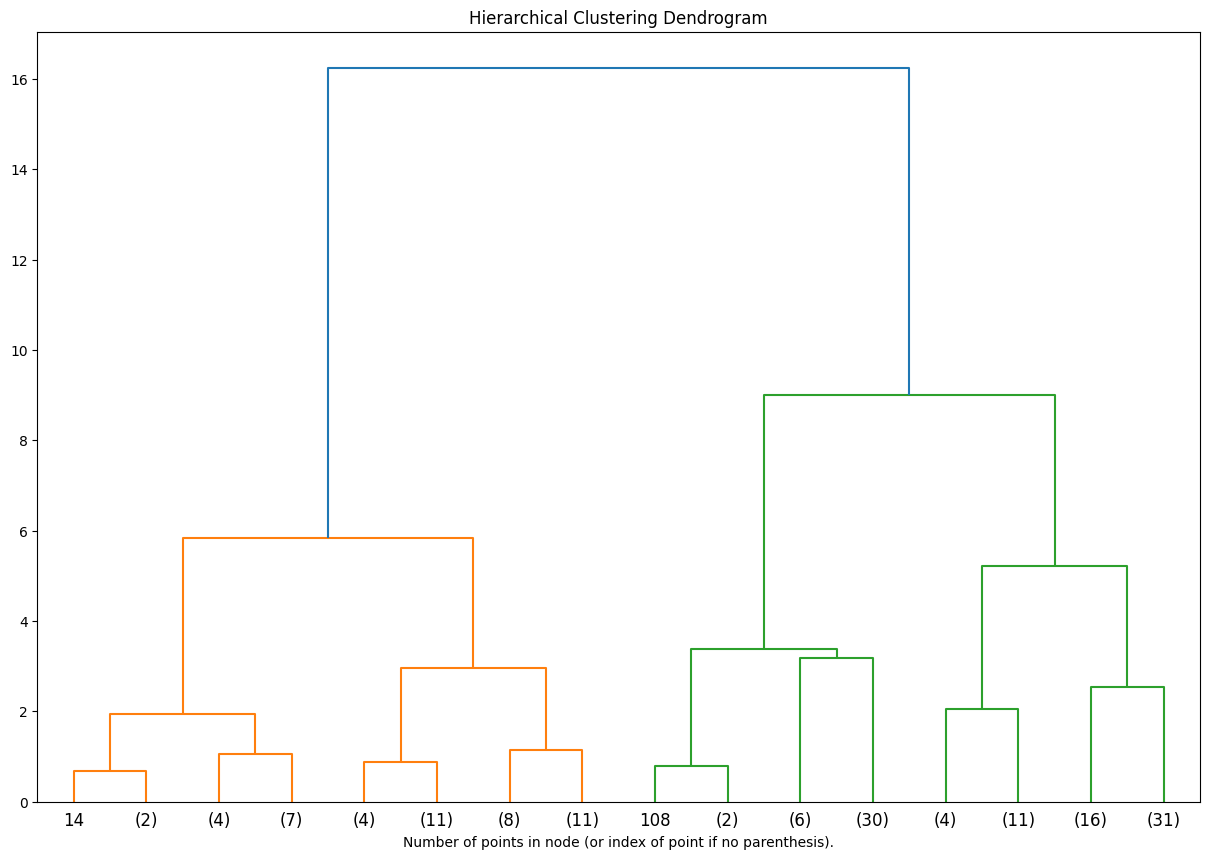


Рисунок 48 – Дендрограмма объединения кластеров в агломеративной кластеризации

Задание на лабораторную работу

* 1. Получить у преподавателя датасет для проведения кластерного анализа.
  2. Осуществить масштабирование данных и перейти в пространство пониженной размерности (2-х или 3-х мерное).
  3. Построить визуализацию метода локтя для определения оптимального числа кластеров для разбиения.
  4. Реализовать алгоритм k-means для найденного в п.2 оптимального числа кластеров. Построить графики получившегося разбиения.
  5. Построить дендрограмму разбиения датасета на кластеры и определить оптимальное число кластеров для разбиения.
  6. Реализовать алгоритм агломеративной кластеризации для найденного в п.4 числа кластеров. Построить графики получившегося разбиения.
  7. Подобрать настраиваемые параметры алгоритма DBScan для разбиения на оптимальное количество кластеров, найденных в п.2 и п.4. Построить графики получившегося разбиения.

Лабораторная работа № 5. Метод наименьших квадратов. Градиентный спуск. Стохастический градиентный спуск.

Теоретическая часть

В данной лабораторной работе будут рассмотрен метод наименьших квадратов с точки зрения его применения к алгоритму линейной регрессии в машинном обучении, а также алгоритм градиентного спуска и его стохастической модификации для решения задачи нахождения минимума функции потерь.

Метод наименьших квадратов.

В лабораторной работе № 2 рассматривался алгоритм линейной регрессии с квадратичной функцией потерь. Рассмотрим, что происходит при подборе параметров модели линейной регрессии и ее построении.

Итак, в простейшем случае, даны две величины *x* и *y,* связанные некоторой табличной зависимостью (полученные экспериментальным путем), представленной на рисунке 49

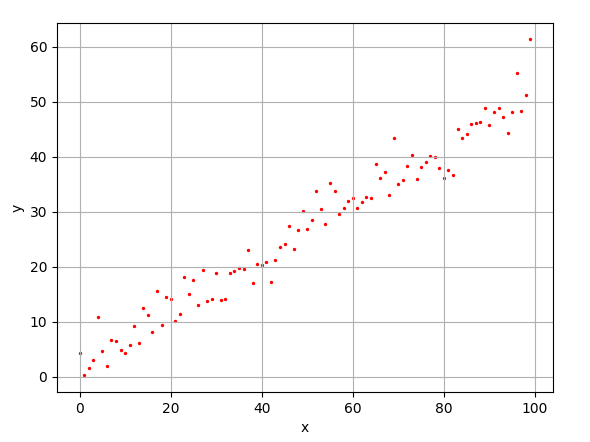


Рисунок 49 – Точки, полученные в результате эксперимента

Требуется установить функциональную зависимость между переменными *x* и *y*по результатам экспериментальных исследований таким образом, чтобы она наиболее полно описывала представленные данные [11]. Например, на рисунке 49 явно прослеживается линейная зависимость. Тогда, для линейной функции (58) необходимо определить два параметра: *k* и *b.*

|  |  |
| --- | --- |
|  | (58) |

Вид функции (58) может быть известен из теоретических соображений или определяться характером расположения экспериментальных точек.

В качестве оценки качества построенной зависимости в методе наименьших квадратов выступает сумма квадратов ошибок отклонений:

|  |  |
| --- | --- |
|  | (59) |

То есть, необходимо найти такие коэффициенты *k* и *b,* чтобы функция потерь (59) была минимальной. Методы решения такой задачи относятся к категории численных методов или методов вычислительной математики. Один из способов аппроксимации функций – интерполяция [12]. В результате решения задачи интерполяции линия, соответствующая интерполирующей функции, будет обязательно проходить через все точки исходных данных.

Применение интерполяции в данном случае нецелесообразно, так как значения *y* в узлах получены экспериментально и поэтому являются сомнительными (данные, получаемые в результате измерений, имеют погрешности). Кроме того, совпадение значений в узлах не означает совпадения характеров поведения исходной и интерполирующей функций. Поэтому необходимо найти такой метод подбора эмпирической формулы, который не только позволяет найти саму формулу, но и оценить погрешность.

Итак, чтобы найти минимум функции потерь (59) необходимо найти первые частные производные по искомым параметрам и приравнять к 0. В итоге, учитывая (58) и (59) получаем следующую систему уравнений:

|  |  |
| --- | --- |
|  | (61) |

Раскроем все суммы и разделим их на *N.* Тогда система (61) примет вид:

|  |  |
| --- | --- |
|  | (62) |

Формулы, получившиеся в суммах, представляют собой первые и второй моменты, а также один смешанный момент:

|  |  |
| --- | --- |
|  | (63) |

При этом знак \* означает, что это экспериментальные моменты. Исходя из (62) и (63) получаем, что:

|  |  |
| --- | --- |
|  | (64) |

Таким образом, получены оценки параметров и для линейной оценки экспериментальных данных по методу наименьших квадратов.

Вернемся к алгоритму линейной регрессии. Функция потерь, которая минимизируется в этом алгоритме имеет вид:

|  |  |
| --- | --- |
|  | (65) |

Таким образом, получаем следующую задачу оптимизации:

|  |  |
| --- | --- |
|  | (66) |

Запишем функцию потерь (66) в матричном виде:

|  |  |
| --- | --- |
|  | (67) |

Продифференцировав функцию (67) по вектору  и приравняв к нулю, можно получить явную аналитическую формулу для решения задачи минимизации:

|  |  |
| --- | --- |
|  | (68) |

Это решение называется *нормальным уравнением* линейной регрессии. Наличие аналитического решения кажется положительным фактором, однако, у него есть некоторые минусы, среди которых вычислительная сложность операции (обращение матрицы  будет иметь кубическую сложность от количества признаков ), а также тот факт, что матрица   может быть вырожденной и поэтому необратимой. Тогда найти решение будет невозможно и необходимо применять различные оптимизационные алгоритмы для нахождения минимума функции потерь.

Градиентный спуск

Учитывая тот факт, что квадратичная функция потерь имеет один минимум и непрерывна на всей области ее значений, то в каждой точке этой функции можно рассчитать частные производные. Градиентом функции называется вектор, показывающий направление наибольшего возрастания функции, а значит антиградиент будет показывать направление наискорейшего убывания:

|  |  |
| --- | --- |
|  | (69) |

Алгоритм градиентного спуска [13] выглядит следующим образом:

1. Задать начальную точку , значение для условия окончания алгоритма и количество итераций *M.*
2. Найти градиент функции потерь в произвольной точке:
3. Положить *k* = 0.
4. Вычислить .
5. Задать величину шага обучения .
6. Вычислить новую точку .
7. Проверить выполнение условия останова . Если условие выполнено, закончить расчет. При этом минимальное значение . При невыполнении условия увеличить значение k на 1 и повторить пункты 4-7.

Применяя алгоритм градиентного спуска к алгоритму линейной регрессии, получаем, что при начальном векторе весов , *k*-й шаг градиентного спуска будет иметь следующий вид:

|  |  |
| --- | --- |
|  | (70) |

При этом, сходимость алгоритма будет определяться на основе следующего критерия:

|  |  |
| --- | --- |
|  | (71) |

Начальный вектор весов  также можно определять различными способами, обычно его берут нулевым или состоящим из случайных небольших чисел.

В случае многомерной регрессии (при количестве признаков больше 1) при оптимизации функционала ошибки (67) формула вычисления градиента принимает вид:

|  |  |
| --- | --- |
|  | (72) |

Однако, использование стандартного алгоритма градиентного спуска, может оказаться нецелесообразным в случае большого признакового пространства решаемой задачи. В этом случае операция произведения матриц и ее транспонирования оказывается слишком ресурсозатратной. В этом случае прибегают к различным адаптациям алгоритма градиентного спуска, в частности к стохастическому градиентному спуску.

Стохастический градиентный спуск

В случае обычного алгоритма градиентного спуска изменение весовых настраиваемых коэффициентов осуществляется на основе выражения (70), при этом выражение градиента в матричной форме выглядит как (72). Если расписать j-ю компоненту этого градиента, то получим:

|  |  |
| --- | --- |
|  | (73) |

Таким образом, чтобы сделать один шаг алгоритма градиентного спуска необходимо провести суммирование по всем объектам обучающей выборки. Поэтому, в случае большого объема такой выборки один шаг алгоритма может потребовать большого объема вычислительных ресурсов.

Стремление к оптимизации процесса привело к появлению *стохастического градиентного спуска* (Stochastic gradient descent, SGD). Идея его основана на том, что на одной итерации предполагается вычитать не вектор градиента, вычисленный по всей выборке, а вместо этого случайно выбирать один объект из обучающей выборки  и вычислять градиент только на этом объекте. То есть градиент только одного слагаемого в функционале ошибки и вычитать именно этот градиент из текущего приближения вектора весов:

|  |  |
| --- | --- |
|  | (74) |

Исходя из этого, можно сказать, что если в обычном алгоритме градиентного спуска график изменения ошибки падает монотонно, то при использовании алгоритма стохастического градиентного спуска уменьшение ошибки происходит на одном объекте, в то время как на другом объекте она может вырасти. Поэтому график изменения ошибки зачастую имеет вид, представленный на рисунке 50.

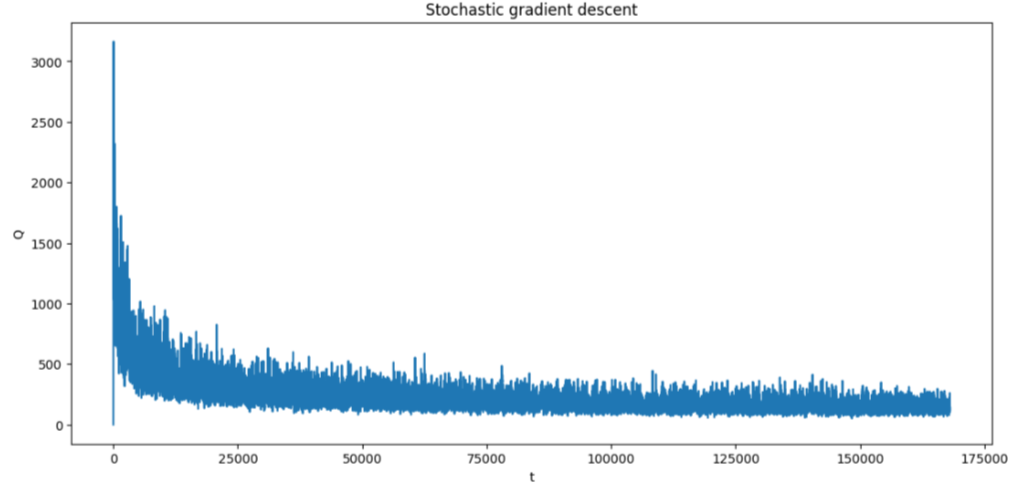


Рисунок 50 – Изменение значения функции потерь в алгоритме стохастического градиентного спуска

При этом, алгоритм стохастического градиентного спуска, за исключением того, что теперь необходимо считать градиент только на одном объекте обучающей выборки отличается от стандартного алгоритма расчетом и изменением текущего функционала качества.

Таким образом, алгоритм стохастического градиентного спуска выглядит следующим образом:

1. Задать начальную точку , значение для условия окончания алгоритма и количество итераций *M.*
2. Найти градиент функции потерь в произвольной точке :
3. Вычислить начальное значение функционала качества .
4. Положить *k* = 0.
5. Вычислить .
6. Задать величину шага обучения .
7. Вычислить новую точку .
8. Пересчитать значение функционала качества по формуле:
9. Проверить выполнение условия останова . Если условие выполнено, закончить расчет. При этом минимальное значение . При невыполнении условия увеличить значение k на 1 и повторить пункты 4-8.

Практическая часть.

Метод наименьших квадратов

Рассмотрим решение задачи нахождения коэффициентов k и b на примере линейной функции:

В таблице 5.1 показан пример зависимости y = f(x).

Таблица 5.1. Экспериментальные данные

|  |  |
| --- | --- |
| x | y |
| 0.28 | 4.14 |
| 0.08 | 3.13 |
| 0.44 | 4.95 |
| 0.47 | 5.09 |
| 0.05 | 2.99 |

Найдем коэффициенты k и b воспользовавшись формулой (64). Получим следующие значения:

k = 4.95

b = 2.75

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Гладилин П.Е., Боченина К.О., Технологии машинного обучения: учебно-методическое пособие – СПб: Университет ИТМО, 2020. – 75 с.
2. Максимова Т.Г., Попова И.Н. Эконометрика: учебно-методическое пособие / Т.Г. Максимова, И.Н. Попова. – СПб.: Университет ИТМО, 2018. – 70 с.
3. Деревья решений. URL: <https://scikit-learn.ru/1-10-decision-trees/> (дата обращения 07.09.2023).
4. А.В. Кугаевских, Д.И. Муромцев, О.В. Кирсанова. Классические методы машинного обучения. – СПб: Университет ИТМО, 2022. – 53 с.
5. Boser B., Guyon I., Vapnik V. A training algorithm for optimal margin classifiers // In Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory. 1992. – P. 144-152.
6. Воронцов К.В. Лекции по методу опорных векторов. ВЦ РАН, Москва. [Электронный источник] URL:http://www.ccas.ru/voron/download/SVM.pdf (дата обращения 28.09.2023).
7. Градиентный бустинг. [Электронный источник] URL: <https://neurohive.io/ru/osnovy-data-science/gradientyj-busting/> (дата обращения 28.09.2023).
8. Thorndike R.L. Who Belongs in the Family? // Psychometrika, 1953, 18 (4): 267–276.
9. Кластеризация. [Электронный источник] URL: <https://academy.yandex.ru/handbook/ml/article/klasterizaciya> (дата обращения 09.10.2023).
10. Hierarchical Clustering // Cluster Analysis. — John Wiley & Sons, Ltd, 2011. — Chap. 4. P. 71–110. — DOI: 10.1002/9780470977811.ch4.
11. Л.В. Коломиец, Н.Ю. Поникарова. Метод наименьших квадратов: метод. указания / сост.: Л.В. Коломиец, Н.Ю. Поникарова. – Самара: Изд-во Самарского университета, 2017. – 32 с.
12. Малышева Т.А. Численные методы и компьютерное моделирование. Лабораторный практикум по аппроксимации функций: Учеб.-метод. пособие. СПб.: Университет ИТМО, 2016. 33 с.
13. Пантелеев А.В. Методы оптимизации в примерах и задачах: Учеб. пособие/А.В. Пантелеев, Т.А. Летова. – 2-е изд., исправл. – М.: Высш. Шк., 2005. – 544 с.: ил.