**МИНИСТЕРСТВО СЕЛЬСКОГО ХОЗЯЙСТВА
РОССИЙСКОЙ ФЕДЕРАЦИИ**

**РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ АГРАРНЫЙ УНИВЕРСИТЕТ -
МСХА имени К.А. ТИМИРЯЗЕВА**

В.В. Демичев, Д.В. Быков, А.С. Невзоров, В.С. Токарев

БОЛЬШИЕ ДАННЫЕ

Учебное пособие

Москва, 2024

УДК 004.42
ББК -018*32.973
Б79

Рецензент:
Сальников С.Г. – руководитель отдела информатизации агропромышленного комплекса Всероссийского института аграрных проблем и информатики им. А.А. Никонова (ВИАПИ) – филиала ФГБНУ ФНЦ ВНИИЭСХ, кандидат физико-математических наук

Демичев В.В., Быков Д.В., Невзоров А.С., Токарев В.С.

Большие данные: учебное пособие / В.В. Демичев, Д.В. Быков, А.С. Невзоров, В.С. Токарев. – М.: Издательство «Научный консультант», 2024. – 86 с.

Б79

ISBN 978-5-907933-16-3

Учебное пособие включает методические указания, теоретические положения, материалы для самостоятельной работы и контроля знаний студентов.

В учебном пособии изложены основные этапы, технологии, алгоритмы и инструменты обработки больших данных, включая высокоуровневый язык программирования Python. В пособии рассматриваются современные подходы к сбору, хранению, исследованию, моделированию и визуальному представлению результатов анализа больших данных.

Предназначено для студентов вузов направления подготовки 09.03.02 «Информационные системы и технологии». Рекомендовано к изданию учебно-методической комиссией института экономики и управления АПК ФГБОУ ВО «РГАУ-МСХА имени К.А. Тимирязева» (протокол № 1 от 30 августа 2024 г.).

УДК 004.42
ББК -018*32.973

ISBN 978-5-907933-16-3

СОДЕРЖАНИЕ

ПРЕДИСЛОВИЕ

Учебное пособие подготовлено в соответствии с требованиями Федерального государственного образовательного стандарта высшего профессионального образования (ФГОС ВО) по направлению подготовки 09.03.02 «Информационные системы и технологии».

Учебное пособие включает методические указания, теоретические положения, материалы для самостоятельной работы и контроля знаний студентов.

В учебном пособии изложены основные этапы, технологии, алгоритмы и инструменты обработки больших данных, включая высокоуровневый язык программирования Python. В пособии рассматриваются современные подходы к сбору, хранению, исследованию, моделированию и визуальному представлению результатов анализа больших данных.

Содержание учебного пособия позволяет вузам вести подготовку специалистов по специальностям в сфере информатики и вычислительной техники на очном, очно-заочном и заочном отделениях. Оно может быть использовано также при подготовке в сельскохозяйственных вузах студентов и аспирантов специальностей, связанных с информационными технологиями.

При написании пособия учтен опыт преподавания дисциплины «Большие данные» ФГБОУ ВО «РГАУ-МСХА имени К.А. Тимирязева» и других вузах России.

Практические задания охватывают все ключевые темы, связанные с теорией и практикой программирования на современных высокоуровневых языках, теорий алгоритмов и структур данных.

В каждом разделе приводятся задачи для самостоятельной работы, в ряде случаев повышенной сложности, что требует от студентов предварительного изучения теоретических вопросов на лекциях, по учебникам и учебным пособиям, рекомендованным кафедрами. Контрольные вопросы и задания используются студентами для самоконтроля, а преподавателями – при промежуточной проверке знаний студентов и проведении итоговых контрольных работ.

Учебное пособие подготовили на кафедре статистики и кибернетики: доцент, кандидат экономических наук В.В. Демичев, ассистенты Д.В. Быков, А.С. Невзоров., Титов А.Д., Токарев В.С.

Раздел 1
Подготовка больших данных для анализа, исследования и моделирования

Лабораторная работа № 1.1
«Очистка и преобразование данных. Определение «выбросов» (outliers) в датасете»

Теоретические положения

Планирование основных этапов анализа больших данных обеспечивает повышение его эффективности и прозрачности для всех заинтересованных сторон. Кроме того, поэтапное описание процесса анализа, позволяет работать над ним проектно, в команде, с привлечением достаточного количества аналитиков и разработчиков. Основные этапы анализа больших данных – постановка цели, сбор данных, подготовка данных, исследование данных, моделирование данных, отображение и автоматизация.

1. Процесс начинается с назначения **цели исследования**. На этом этапе необходимо четко понимать «что» необходимо сделать, «как» и «почему». Этот этап является важным моментом для разработки, в конечном итоге, всего технического задания.
2. На втором этапе осуществляется **сбор данных**. Проведение качественного исследования требует сбора данных из всех доступных для авторов исследования источников. На этом этапе данные формируются в таблицах Excel и баз данных.
3. Собранные необработанные данные необходимо **подготовить.** На этом этапе данные из низкоуровневой формы преобразуются в данные, которые могут напрямую использоваться в ваших моделях. Для этого в данных выявляются и исправляются всевозможные ошибки, данные из различных источников объединяются и преобразуются.
4. На четвертом этапе выполняется **исследование данных.** Конечной целью. Этого этапа является глубокое понимание данных. Осуществляется поиск закономерностей, корреляций и отклонений, основанных на визуальных и описательных методах.
5. **Построение модели.** На данном этапе осуществляется построение моделей реализации поставленных в исследовании целей – прогнозирования, классификации, кластеризации и других. Модели могут быть достаточно сложными, например, модели машинного обучения. На практике, можно встретить случаи, когда совокупность простых моделей, допустим, моделей линейной регрессии показывают большую эффективность.
6. На данном этапе осуществляется **отображение и автоматизация полученных результатов.** Очень часто на данном этапе разрабатывается веб-приложение для дальнейшей автоматизации и отображения результатов статистического анализа. Также на данном этапе демонстрируются и

интерпретируются полученные результаты, в том числе касательно наиболее важных выводов относительно предметной области исследования.

В таком виде результаты могут быть представлены заказчику и пользователю. Конечно, результаты могут быть представлены как в виде презентации, так и в виде научно-исследовательского отчета. Однако, создание веб-приложения позволяет автоматизировать производимый анализ, с возможностью его дальнейшего развития и углубления. Таким образом, результаты анализа, модели могут быть применены в другом проекте или задействованы в рабочем процессе при изменении или обновлении набора данных.

Представленные этапы не являются строго линейными в своем исполнении и зачастую носят итеративный характер, что означает возможность возвращения и корректировки каждого из этапов.

Подготовка данных состоит из множества аспектов, работа над которыми существенно облегчит этап моделирования.

Очистка, интеграция и преобразование данных. Основная задача здесь – убрать дефекты и подготовить данные для использования в фазах моделирования и представления результатов. Это очень важный момент, потому что ваши модели будут работать лучше и вы потратите меньше времени на исправление аномальных результатов. К сожалению, в моделировании все действует по принципу: «мусор на входе - мусор на выходе». Ваша модель должна получать данные в конкретном формате, так что преобразование данных всегда будет играть важную роль. Привыкайте исправлять ошибки в данных на как можно более ранней стадии процесса. Впрочем, в реальной ситуации это не всегда возможно, так что вам придется вносить исправления в свою программу.

Очистка данных представляет собой подпроцесс направленный на устранение ошибок в данных с тем, чтобы эти данные адекватно и последовательно представляли процесс, в результате которого они были получены. «Адекватное и последовательное представление» означает, что существует как минимум два типа ошибок. К первому типу относятся ошибки интерпретации, когда вы принимаете на веру значение в данных (пример: из данных следует, что возраст человека превышает 300 лет). Ошибки второго типа связаны с расхождениями между источниками данных или стандартизированными значениями. Например: в одной таблице денежные суммы хранятся в рублях, в другой – в долларах.

Ошибки ввода данных. Процессы сбора и ввода данных подвержены ошибкам. Они часто требуют человеческого участия, а поскольку люди не идеальны, они допускают опечатки или отвлекаются и вносят ошибки в технологическую цепочку. Впрочем, данные, собранные машинами или компьютерами, тоже не застрахованы от ошибок. Одни ошибки появляются из-за человеческого несовершенства, другие обусловлены сбоями машин или оборудования. В частности, ко второй категории относятся ошибки, происходящие из ошибок передачи данных или ошибок в фазах извлечения,

преобразования и загрузки. Также могут встречаться такого рода ошибки как избыточные пробелы, расхождение в регистре символов, невозможные значения, отсутствующие значения, разные единицы измерения, разные уровни агрегирования, выбросы.

Выбросы (outliers). Выбросом называется результат наблюдений, заметно отклоняющийся от других результатов, или более конкретно - результат наблюдений, который обусловлен иной логикой или иным порождающим процессом, чем другие результаты. Простейший способ поиска выбросов основан на использовании диаграмм или таблиц с минимумами и максимумами (Рисунок 1.1).

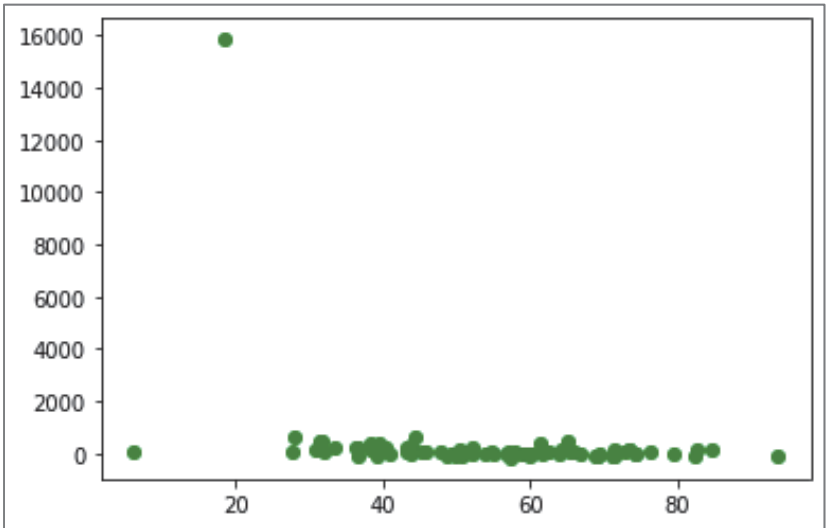


Рисунок 1.1 Диаграмма рассеивания (зависимости) рентабельности субсидий (%) от удельного веса животноводства в регионе (%)

На рисунке 1.1 представлена диаграмма рассеяния с примером наличия выбросов в данных, существенно искажающих результаты построения моделей.

Комбинирование данных. Существует две операции комбинирования данных – соединение таблиц с данными из разных источников. Вторая операция – дополнение таблицы данными из другой таблицы. Все это приводит к расширению данных, участвующих в анализе и исследовании. Также расширение данных может осуществляться на основе расчета новых показателей, например, относительных показателей на основе ранее собранных абсолютных показателей. Однако, существуют ситуации, когда количество показателей становится слишком большим и часть показателей необходимо удалить.

Преобразование данных может осуществляться в форме логарифмирования переменных, что существенно снижает их вариацию и представляет в более линейной форме. Также преобразование данных может проводиться на основе замены качественных величин на количественные. Например, наличие у сотрудника высшего образования. Ответ «да, имеется», преобразуется в фиктивную переменную (dummy variables) – 1, ответ «нет, отсутствует» – 0.

Задание:

Условие. Имеются данные о продаже товаров в продуктовом интернет-магазине (файл «Customers»).

Требуется загрузить данные в Google Colab или среду Spyder, «почистить» данные (если требуется), рассчитать описательную статистику (Descriptive statistics), построить гистограмму по переменной «Prices». Сделать выводы.

Данные: https://disk.yandex.ru/d/SKOuU856U8TJUw

Методические указания к выполнению лабораторной работы

1. На первом шаге загрузим данные в среду разработки и выведем таблицу с данными:

```
#выберем файл для загрузки
from google.colab import files
uploaded = files.upload()
```

Выберем файл на компьютере и загрузим его. В результате появится информация о загрузке файла:

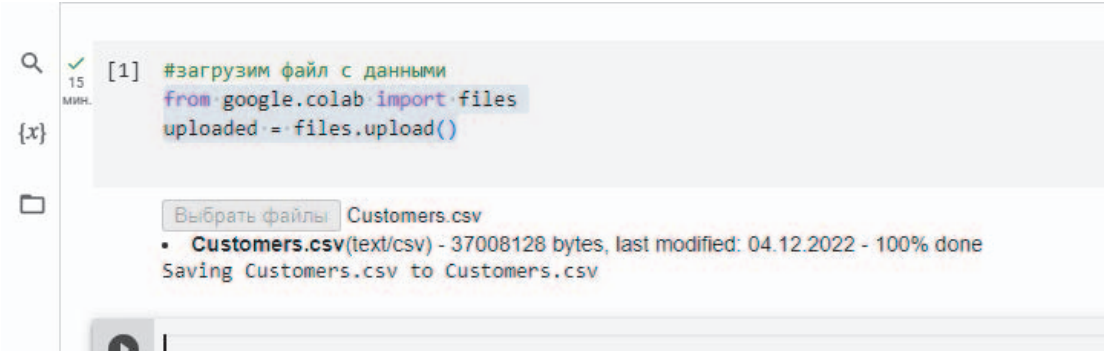
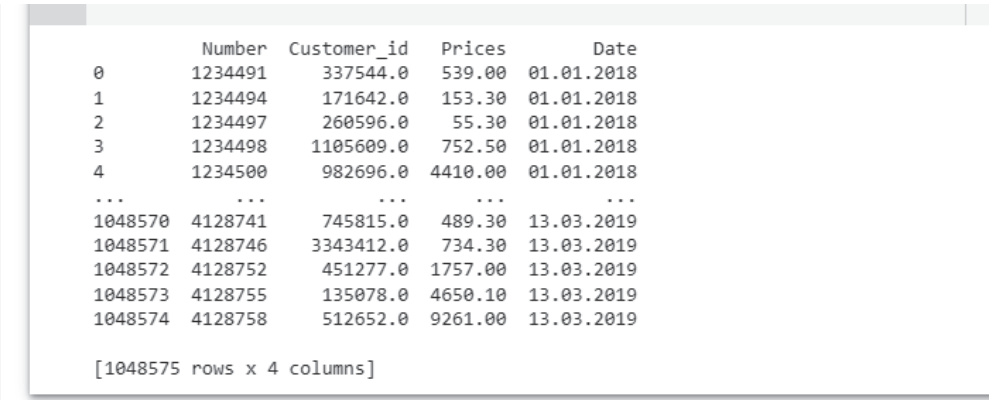


Рисунок 1.2 Информация о загрузке файла в Google Colab

Далее напишите следующий фрагмент кода, чтобы импортировать его во фрейм данных pandas (импортируем все необходимые библиотеки):

```
import io
import pandas as pd
df = pd.read_csv(io.BytesIO(uploaded['file.csv']), sep=";", decimal=",")
    print(df)
```



	Number	Customer_id	Prices	Date
0	1234491	337544.0	539.00	01.01.2018
1	1234494	171642.0	153.30	01.01.2018
2	1234497	260596.0	55.30	01.01.2018
3	1234498	1105609.0	752.50	01.01.2018
4	1234500	982696.0	4410.00	01.01.2018
...	...	...	...	...
1048570	4128741	745815.0	489.30	13.03.2019
1048571	4128746	3343412.0	734.30	13.03.2019
1048572	4128752	451277.0	1757.00	13.03.2019
1048573	4128755	135078.0	4650.10	13.03.2019
1048574	4128758	512652.0	9261.00	13.03.2019

[1048575 rows x 4 columns]

Рисунок 1.3 Результат импортирования данных во фрейм данных pandas

Благодаря вызову метода print() мы можем увидеть данные, которые были подгружены в Google Colab. Таким образом, мы имеем 4 столбца в таблице и более 1 млн. строк.

2. Далее проведем некоторый анализ этих данных на предмет их формата и типа данных, а также определение числовых и нечисловых данных:

#выведем формат и тип данных

```
print(df.shape)
print(df.dtypes)
```

Результат:

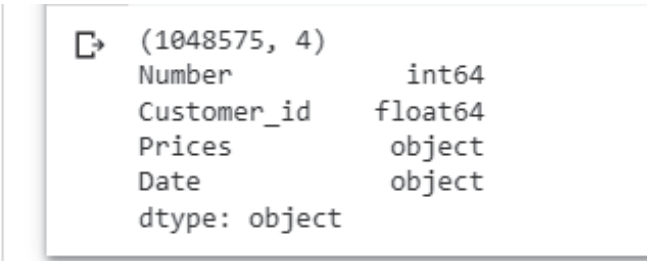


Рисунок 1.4 Определение типа и формата данных

По выведенным значениям видно, что колонки Number и Customer_id являются числовыми, а Date и Prices является нечисловой. По нечисловым данным описательная статистика не рассчитается, поэтому переведем столбец «Prices» в числовой вид с помощью метода .astype(float):

```
#преобразуем данные в столбце Prices
df = df[['Prices', 'Number', 'Customer_id']].astype(float)
```

Колонкой дата в нашем задании можно пренебречь. После этого посмотрим какие колонки в таблице относятся к численным значениям, а какие нет:

#выберем числовые данные. Для этого нам потребуется добавить библиотеку numpy.

```
import numpy as np
# выберем числовые колонки
df_numeric = df.select_dtypes(include=[np.number])
numeric_cols = df_numeric.columns.values
print(numeric_cols)
# выберем нечисловые колонки
df_non_numeric = df.select_dtypes(exclude=[np.number])
non_numeric_cols = df_non_numeric.columns.values
print(non_numeric_cols)
```

И увидим, что требующаяся нам для колонка с ценой находится в числовом типе.

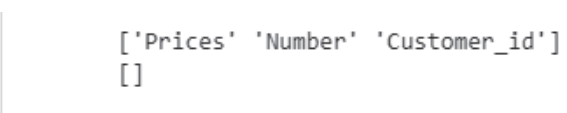


Рисунок 1.5 Списки числовых и нечисловых значений в Python

3. Далее проведем описательную статистику:

```
#Описательная статистика
print(df.describe())
```

Результат:

	Prices	Number	Customer_id
count	1.048575e+06	1.048575e+06	1.048566e+06
mean	2.136475e+03	2.570549e+06	1.485835e+06
std	3.243518e+03	8.434932e+05	8.833536e+05
min	-3.430000e+01	1.234491e+06	0.000000e+00
25%	7.049000e+02	1.804274e+06	7.050775e+05
50%	1.358000e+03	2.524455e+06	1.438339e+06
75%	2.422000e+03	3.291542e+06	2.124041e+06
max	8.190966e+05	4.128758e+06	3.343412e+06

Рисунок 1.6 Описательная статистика

Из полученной описательной статистики можно сделать вывод, что максимальная и минимальная стоимости равны соответственно 819097 руб. и -34 руб. (отрицательное значение цены должно насторожить исследователя), средняя цена равна 2137 руб., а стандартное отклонение цены равно 3243 руб.

4. Следующим шагом проверим таблицу на отсутствующие значения:

```
#Тепловая карта отсутствующих данных (Missing Data Heatmap)
#Добавляем библиотеку seaborn
import seaborn as sns
cols = df.columns[:4] # выбираем первые 4 колонки
colours = ['#000099', '#ffff00'] # определите цвета – желтые – это пропущенные. синие – не пропущенные.
sns.heatmap(df[cols].isnull(), cmap=sns.color_palette(colours))
```

Рисунок 1.7 Визуализация пропущенных значений

Визуальный анализ не показал наличия пропущенных значений. Поэтому далее проведем численный анализ.

5. Численный анализ пропущенных значений:

```
#Процент пропущенных данных(Missing Data Percentage List)
# % пропущенных данных
for col in df.columns:
   pct_missing = np.mean(df[col].isnull())
   print('{} - {}%'.format(col, pct_missing*100))
```

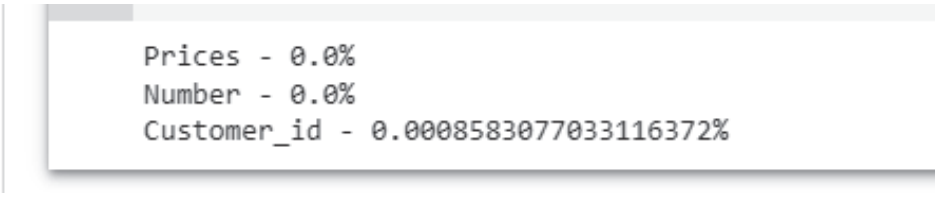


Рисунок 1.8 Численный анализ пропущенных значений

Обнаружено, что у 0.0009% покупок не прописан Customer_id покупателя. Так как данный показатель не является критическим, а остальные показатели в строке присутствуют, заменим пропущенные значения медианным, равным 1438339:

```
#Решение проблемы пропущенных данных.
# заменим пропущенные значения медианой
median = df['Customer_id'].median()
print(median)
df['Customer_id'] = df['Customer_id'].fillna(median)
```

Проверка:

```
#Проверка пропущенных данных после замены пропущенных значений медианой
for col in df.columns:
   pct_missing = np.mean(df[col].isnull())
   print('{} - {}%'.format(col, pct_missing*100))
```

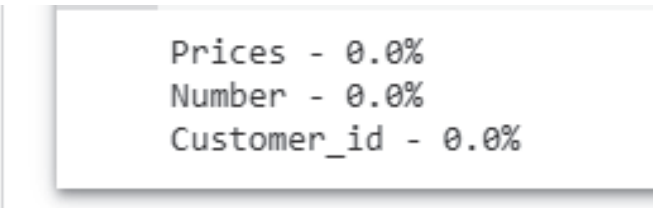


Рисунок 1.9 Численный анализ пропущенных значений после замены

Все пропущенные значения устранены.

6. Далее визуализируем разброс значений по переменной Prices, используя диаграмму рассеяния

```
#Выявление выбросов

import matplotlib.pyplot as plt
```

```
#Визуализация разброса значений по Prices
fig, ax = plt.subplots(figsize=(10, 6))
ax.scatter(x = df['Customer_id'], y = df['Prices'])
plt.xlabel("Customer_id")
plt.ylabel("Prices")
plt.show()
print(df.Prices.mean())
print(df.head())
```

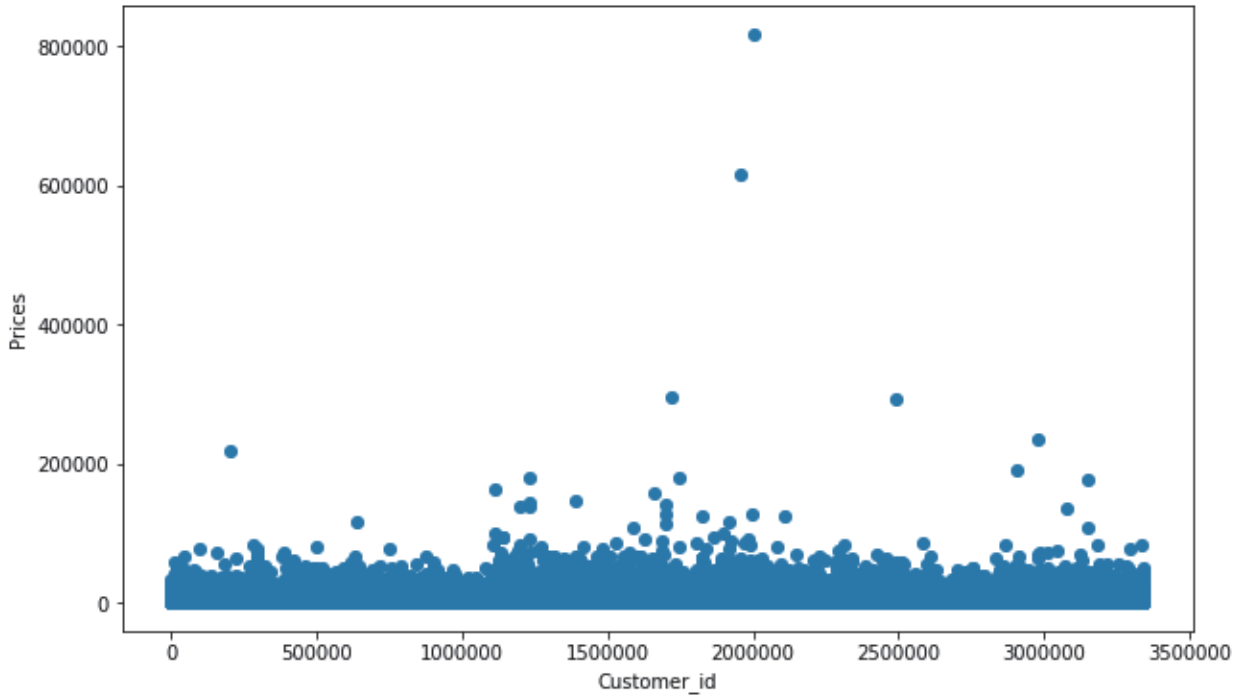


Рисунок 1.10 Диаграмма рассеяния

Из диаграммы следует, что в совокупности присутствуют выпадающие значения, которые необходимо удалять или заменять средними значениями (или медианами).

Самостоятельная работа

1. Удалите строки с выбросами признака “Prices”. Постройте диграмму рассеяния после удаления данных строк.
2. Выведите все заказы стоимостью больше 10 млн. руб.
   Имя файла[имя файла['price'] > значение]
3. Удалить все покупки стоимостью выше 100 руб.
   Имя файла[~(имя файла['price'] > значение)]
4. Необходимо увеличить признак «Prices» на 20%.

Имя файла['имя столбца'] = имя файла['имя столбца'].apply(lambda x:x*1.2)

Вопросы для самоконтроля

1. Какая библиотека Python применяется для работы с данными в части их загрузки в среду и удаления резко отличающихся значений?
2. Как называется совокупность показателей, позволяющих исследуемую совокупность?
3. Назовите основные подходы к выявлению пропущенных значений в совокупности.
4. Для чего применяется диаграмма рассеяния на этапе подготовки и очистки данных?
5. Библиотека Python для построения диаграммы рассеяния.

Лабораторная работа № 1.2
«Исследование данных»

Теоретические положения

На этапе исследовательского анализа происходит углубленное изучение данных. В графическом формате информация воспринимается намного проще, поэтому исследовательский этап реализуется посредством построения всевозможных графиков. Цель – наиболее полно узнать тот набор данных, который будет использован на этапе моделирования. Применяется широкий спектр методов визуализации, от простых графиков или столбиковых диаграмм, до более сложных диаграмм Сэнки и сетевых графов. Например, построение коррелограммы или графика автокорреляции позволяет понят присутствует во временных рядах тренд или нет (Рисунок 1.11).

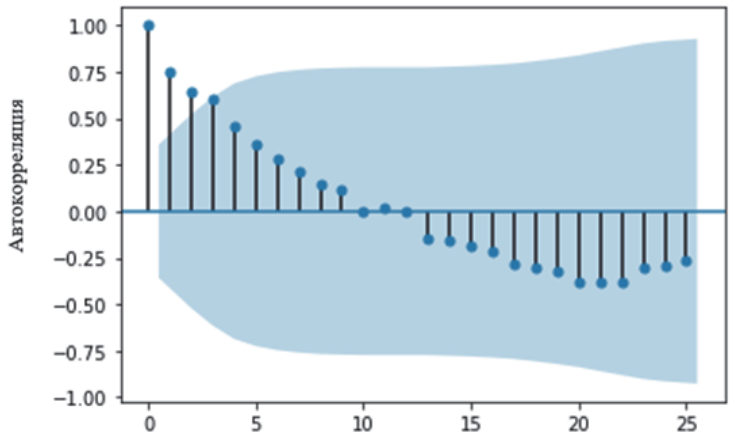


Рисунок 1.11 Графическое представление автокорреляционной функции

На практике все не ограничивается методами визуализации. Частью исследовательского анализа могут быть кластеризация, классификация и

другие методы.

Задание:

Условие. Имеются данные о продаже товаров в продуктовом интернет-магазине (файл «Customers»).

Требуется загрузить данные в Google Colab или среду Spyder, построить по признаку «Prices» линейный график, гистограмму, а также график автокорреляции. Сделать выводы.

Данные: https://disk.yandex.ru/d/SKOuU856U8TJUw

Методические указания к выполнению лабораторной работы

1. Стоит отметить, что после загрузки данных в среду:

```
import pandas as pd
df = pd.read_csv('/Customers.csv', sep=";", decimal=",")
```

И вывода датасета в консоли или в среде:

```
df
```

	Number	Customer_id	Prices	Date
0	1234491	337544.0	539.00	01.01.2018
1	1234494	171642.0	153.30	01.01.2018
2	1234497	260596.0	55.30	01.01.2018
3	1234498	1105609.0	752.50	01.01.2018
4	1234500	982696.0	4410.00	01.01.2018
...	...	...	...	...
1048570	4128741	745815.0	489.30	13.03.2019
1048571	4128746	3343412.0	734.30	13.03.2019
1048572	4128752	451277.0	1757.00	13.03.2019
1048573	4128755	135078.0	4650.10	13.03.2019
1048574	4128758	512652.0	9261.00	13.03.2019

1048575 rows × 4 columns

Рисунок 1.12 Вывод датасета в среде Google Colab

Среда предлагает представить данную таблицу в интерактивном виде (синие иконки справа вверху рисунка 1.12), а также построить автоматически ряд наиболее подходящих для набора данных графиков.

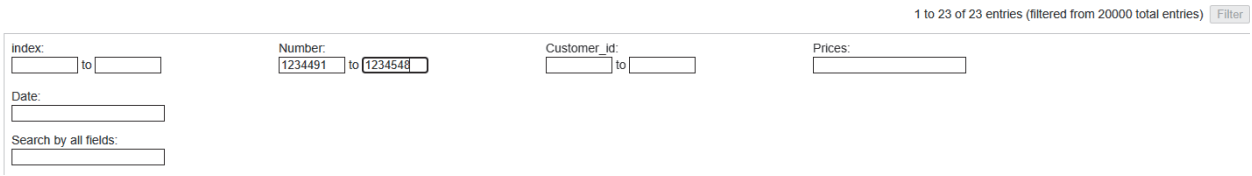


Рисунок 1.13 Возможности фильтра интегративной таблицы

В интерактивную таблицу встроен фильтр, позволяющий просматривать данные по определенным, заданным параметрам.
Также Google Colab автоматически предлагает ряд графиков для некоторых показателей датасета.

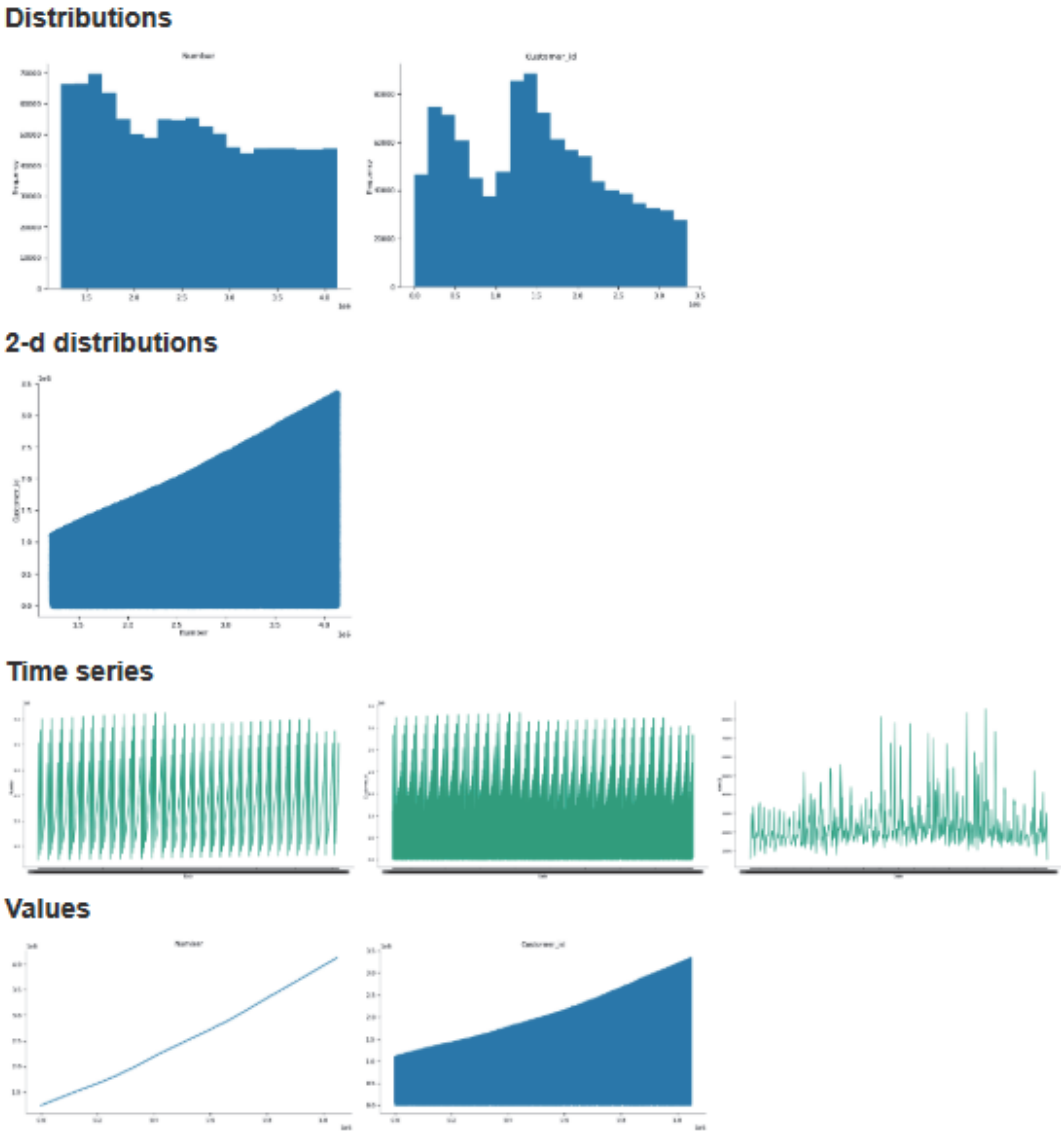


Рисунок 1.14 Автоматический вывод графиков в среде Google Colab

Мы видим, что средой построены графики распределения и временных рядов, которые могут быть сохранены. Однако, средой не построены необходимые нам гистограмма и линейный график по признаку «Prices». Нажав, на рисунок, среда покажет нам код, с применением которого были построены данные графики.

```
from matplotlib import pyplot as plt
df['Customer_id'].plot(kind='hist', bins=20, title='Customer_id')
plt.gca().spines[['top', 'right',]].set_visible(False)
```

Рисунок 1.15 Скрипт для построения гистограммы

Скопировав код и заменив признак 'Customer_id' на 'Prices', используем его для построения гистограммы по требуемому признаку.

В случае возникновения проблем с нечисловыми данными помним о возможности их преобразования в числовой формат:

df = df[['Prices', 'Number', 'Customer_id']].astype(float)

Обратите внимание, что возможно потребуется изменение параметра bins в методе .plot() иначе все имеющиеся данные попадут в один интервал данных.

Также из описательной статистики мы знаем, что 75% покупок не превышало 2500 руб. Поэтому, для исключения резко отличающихся значений мы можем ограничить датасет покупками не превышающими данную сумму.

df = df[df['Prices'] < 100000]

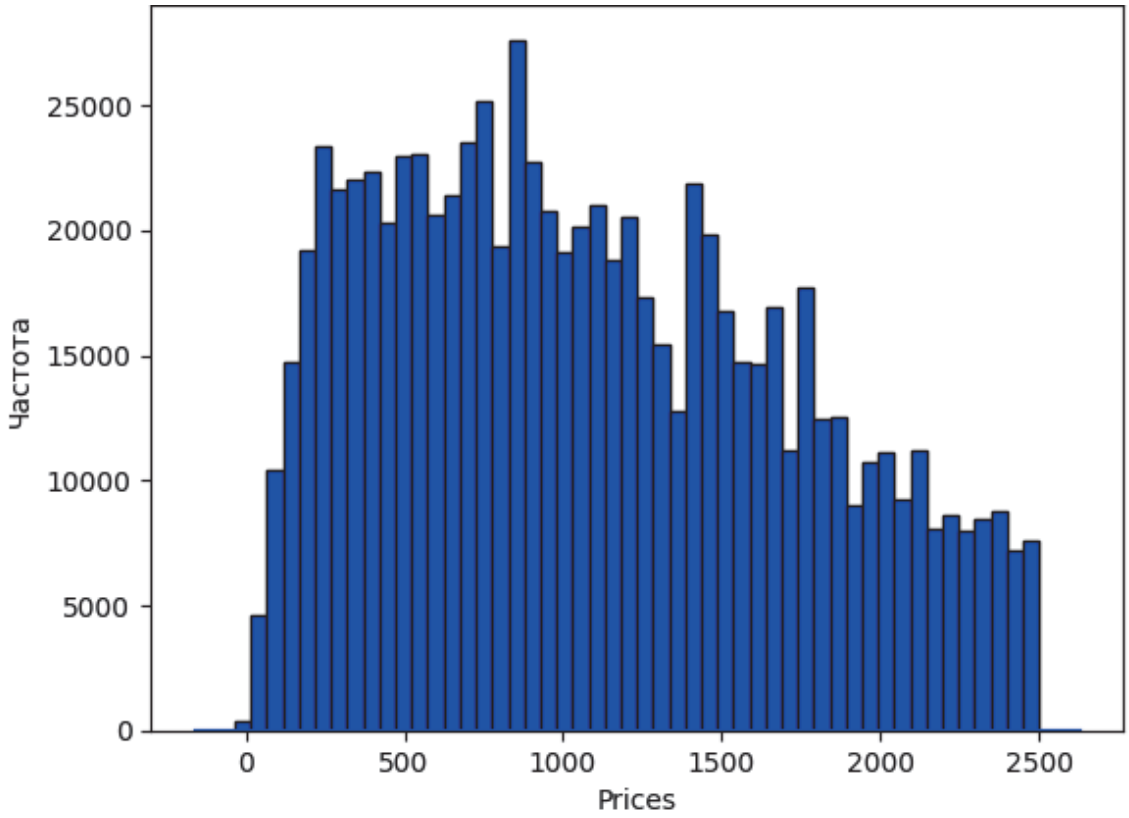


Рисунок 1.15 Гистограмма распределения цен покупок в интернет-магазине в диапазоне цен до 2500 руб.

В представленном диапазоне цен мы видим неравномерное распределение и преобладание покупок в пределах 1000 руб.

Второй вариант построения гистограммы может быть реализован в виде:

```
# Импортируем библиотеки
import matplotlib.pyplot as plt
import seaborn as sns
# matplotlib гистограмма
```

```
plt.hist(df['Prices'], color = 'blue', edgecolor = 'black',
         bins = 50)

# seaborn гистограмма
sns.distplot(df['Prices'], hist=True, kde=True,
             bins=50, color = 'blue',
             hist_kws={'edgecolor':'black'})
# Добавим подписи данных
plt.title('Гистограмма распределения цен покупок в интернет-магазине')
plt.xlabel('Prices')
plt.ylabel('Частота')
```

2. Далее построим линейный график по признаку 'Prices' для первых 100 покупок.

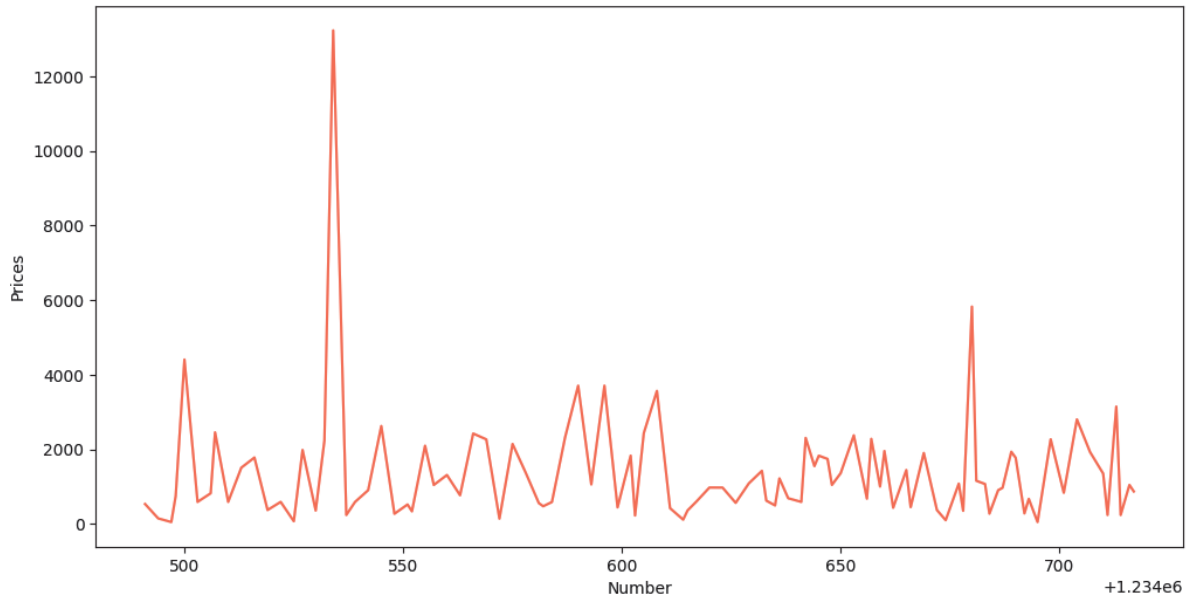


Рисунок 1.16 График продаж для первых 100 покупателей

По линейному графику видно, что вариация признака однородна и имеются два выпада – один выше 12 тыс. руб. второй – в районе 6 тыс. руб.

Реализован рисунок был следующим образом:

```
df = df[:100]
import matplotlib.pyplot as plt
plt.figure(figsize=(12,6))
plt.plot(df["Number"], df["Prices"], color="tomato")
## подпишем оси
plt.xlabel("Number")
plt.ylabel("Prices")
plt.yticks()
```

plt.title("Величина покупок для первых 100 покупателей", loc="left", fontdict=dict(fontsize=20, fontweight="bold"), pad=10)
plt.show()

3. Теперь построим график автокорреляционной функции:

```
#Импортируем функцию autocorrelation_plot и построим график
from pandas.plotting import autocorrelation_plot
autocorrelation_plot(df['Prices'])
pyplot.show()
```

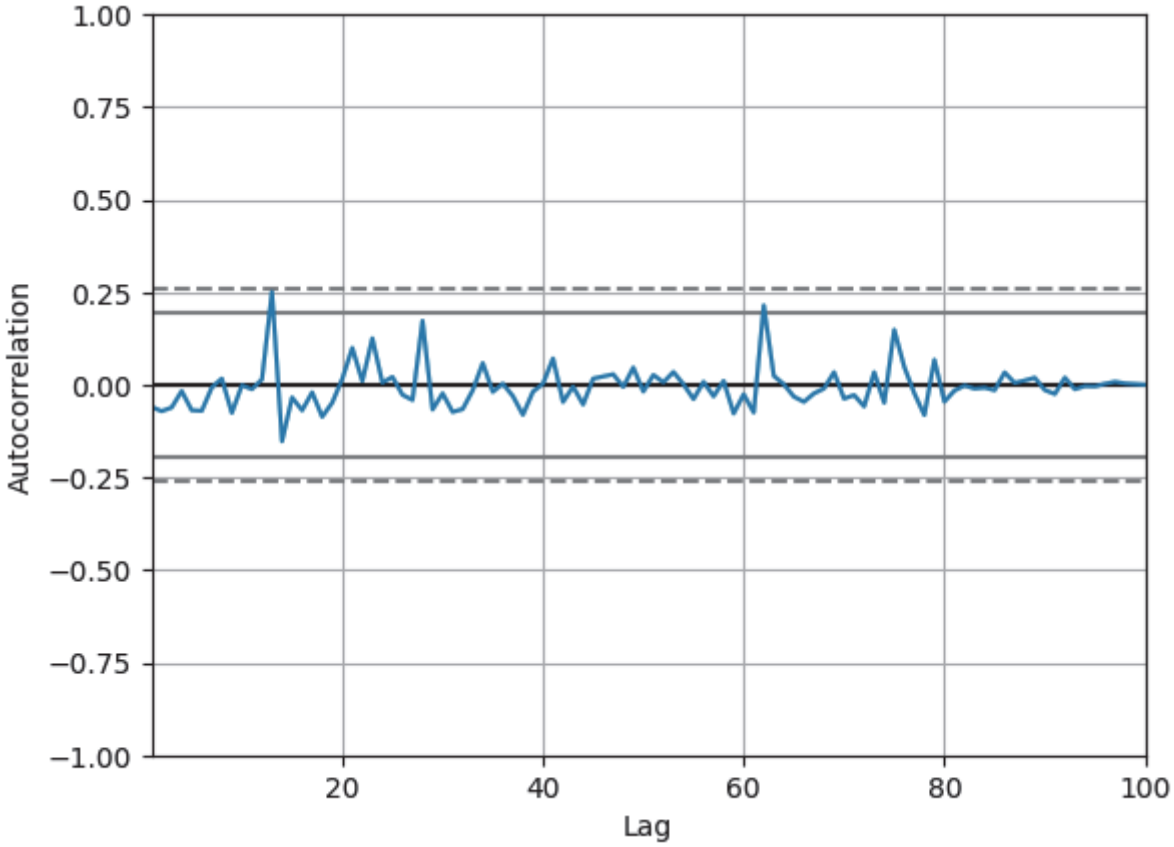


Рисунок 1.17 Автокорреляционная функция

Автокорреляционная функция, представленная на графике, показывает достаточно низкие значения коэффициентов автокорреляции. Кроме того, никаких зависимостей не прослеживается. Не фиксируется явного наличия линейной зависимости или циклических проявлений, кроме наличия некоторых пиков при лагах 15, 25, 65, 75.

Самостоятельная работа

1. Постройте матрицу коэффициентов корреляции между параметром «Prices» и лаговой переменной.

```
#Импортируем необходимые функции и библиотеки
from pandas import DataFrame
from pandas import concat
```

```
#Преобразуем данные и зададим переменные для построения корерляционной матрицы
values = DataFrame(df["Prices"].values)
dataframe = concat([values.shift(1), values], axis=1)
dataframe.columns = ['t-1', 't+1']
#Построим матрицу и выведем результат
result = dataframe.corr()
print(result)
```

2. Постройте автокорреляционную функцию в виде столбиковой диаграммы:

```
#Импортируем функцию plot_acf и построим столбиковую диаграмму
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(df["Prices"], lags=25)
pyplot.show()
```

Вопросы для самоконтроля

1. В чем заключается этап «Исследование данных»?
2. Какие возможности по исследованию данных предоставляет среда Google Colab?
3. Какие выводы могут быть получены при построении гистограммы, линейного графика и автокорреляционной функции?
4. Какой метод Python применяется для построения матрицы автокорреляции?
5. Какие Python библиотеки были использованы на этапе «Исследование данных»?

Лабораторная работа № 1.3
«Кластерный анализ»

Теоретические положения

Кластерный анализ представляет собой метод анализа данных, используемый для поиска групп, объединенных общими атрибутами (также называется группировкой).

Кластерный анализ выполняет следующие основные задачи:

1) Разработка типологии или классификации.

2) Исследование полезных концептуальных схем группирования объектов.

3) Порождение гипотез на основе исследования данных.

4) Проверка гипотез или исследования для определения, действительно ли типы (группы), выделенные тем или иным способом, присутствуют в имеющихся данных.

Независимо от предмета изучения применение кластерного анализа предполагает следующие этапы:

1) Отбор выборки для кластеризации.

2) Определение множества переменных, по которым будут оцениваться объекты в выборке.

3) Вычисление значений той или иной меры сходства между объектами.

4) Применение метода кластерного анализа для создания групп сходных объектов.

5) Проверка достоверности результатов кластерного решения.

Кластерный анализ предъявляет следующие требования к данным: во-первых, показатели не должны коррелировать между собой; во-вторых, показатели должны быть безразмерными; в-третьих, их распределение должно быть близко к нормальному; в-четвёртых, показатели должны отвечать требованию «устойчивости», под которой понимается отсутствие влияния на их значения случайных факторов; в-пятых, выборка должна быть однородна, не содержать «выбросов». Если кластерному анализу предшествует факторный анализ, то выборка не нуждается в «ремонте» – изложенные требования выполняются автоматически самой процедурой факторного моделирования. В противном случае выборку нужно корректировать.

Общепринятой классификации методов кластеризации не существует, но можно выделить ряд подходов.

Наиболее популярный метод кластеризации – это метод k-средних. Действие алгоритма таково, что он стремится минимизировать суммарное квадратичное отклонение точек кластеров от центров этих кластеров:

$$V = \sum_{i=1}^{k} \sum_{x_j \in S_i} \left(x_j - \mu_i\right)^2 \quad (1.1)$$

где k– число кластеров;

S_i – полученные кластеры;

i=1, 2, …, k и μ_i – центры масс векторов $x_j \in S_i$.

Основная идея заключается в том, что на каждой итерации перевычисляется центр масс для каждого кластера, полученного на предыдущем шаге, затем векторы разбиваются на кластеры вновь в соответствии с тем, какой из новых центров оказался ближе по выбранной метрике.

Алгоритм завершается, когда на какой-то итерации не происходит изменения центра масс кластеров. Это происходит за конечное число итераций, так как количество возможных разбиений конечного множества конечно, а на каждом шаге суммарное квадратичное отклонение V уменьшается, поэтому зацикливание невозможно [14].

Другим алгоритмом объединения является метод одиночной связи (метод ближнего соседа). На первом шаге объединяются два наиболее близких объекта, т.е. имеющие максимальную меру сходства. На следующем шаге к ним присоединяется объект с максимальной мерой сходства с одним

из объектов кластера, т.е. для его включения в кластер требуется максимальное сходство лишь с одним членом кластера. Расстояние между двумя кластерами определяется как расстояние между двумя наиболее близкими объектами в различных кластерах.

Еще одним алгоритмом объединения является метод полных связей (метод удаленного соседа). Он позволяет устранить недостаток, присущий методу одиночной связи. Суть правила в том, что два объекта, принадлежащих одной и той же группе (кластеру), имеют коэффициент сходства, который больше некоторого порогового значения S. В терминах евклидова расстояния это означает, что расстояние между двумя точками (объектами) кластера не должно превышать некоторого порогового значения d.

Таким образом, d определяет максимально допустимый диаметр подмножества, образующего кластер. При достаточно большом пороговом значении d расстояние между кластерами определяется наибольшим расстоянием между любыми двумя объектами в различных кластерах [14].

Функция близости – это характеристика кластера, на сколько близки показатели по ряду признаков.

Выделяют следующие основные меры близости:

1) Евклидово расстояние – наиболее общий тип расстояния. Является геометрическим расстоянием между точками в многомерном пространстве:

$$\Phi_{\text{ек}} = \sqrt{\sum_{i=1}^{m}(tik - til)^2} \quad (1.2)$$

2) Квадрат Евклидова расстояния – используется, чтобы придать большие веса более отдельным друг от друга объектам:

$$\Phi_{\text{ек}} = \sum_{i=1}^{m}(tik - til)^2 \quad (1.3)$$

3) Манхэттенское расстояние – по сравнению с евклидовым расстоянием влияние отдельных больших разностей (выбросов) уменьшается, так как они не возводятся в квадрат:

$$\Phi_{\text{ек}} = \sum_{i=1}^{\max n}(til - tik) \quad (1.4)$$

В своем исследование мы используем метод К-средних, в качестве алгоритма объединения, так как он является наиболее простым, что означает высокую скорость выполнения и эффективность по сравнению с другими алгоритмами, в особенности при работе с крупными наборами данных.

Метод к-средних может использоваться для предварительного разбиения на группы большого набора данных, после которого проводится более мощный кластерный анализ подкластеров. Также он может использоваться, чтобы определить количество кластеров и проверить наличие неучтенных данных и связей в наборах.

В качестве функции близости, мы используем Евклидово расстояние, поскольку это самый распространенный метод и может вычисляться по исходным, а не по нормированным данным.

Задание:

Условие. Имеются данные об эффективности государственной поддержки сельского хозяйства регионов Российской Федерации (файл «Clusters»). Показатели представлены в формате Хномер показателя.

Требуется загрузить данные в среду Spyder или Google Colab, определить количество кластеров, определить количество кластеров, рассчитать средние величины показателей по кластерам, сохранить результаты в файл Excel. Сделать выводы.

Данные: https://disk.yandex.ru/d/LtxBREHLiwlauQ

Методические указания к выполнению лабораторной работы

Для реализации модуля кластерного анализа в Python были отобраны показатели эффективности государственной поддержки сельского хозяйства.

Загружаем данные по 78 субъектам Российской Федерации в Python из файла “Clusters” и все необходимые библиотеки для дальнейшего анализа (Рисунок 1.18).

```
#Загрузим данные
from pandas import read_csv
import matplotlib.pyplot as plt
import numpy as np
from scipy.spatial import distance as sci_distance
from sklearn import cluster as sk_cluster
data = read_csv('Clusters.csv',';')
data.head()
```

Рисунок 1.18 Загрузка данных и библиотек для проведения кластерного анализа в Python

Кластеризация проводится методом k-средних, который по сути является алгоритмом *неконтролируемого обучения,* который ищет закономерности в данных на основе сходства. Алгоритм k-средних является одним из простейших и популярных способов группирования данных [14].

В начале нам необходимо определить количество кластеров, по которым будут разделяться входные данные (Рисунок 1.19).

```
#определение количества кластеров
cdata = data
K = range(1, 20)
KM = (sk_cluster.KMeans(n_clusters=k).fit(cdata) for k in K)
centroids = (k.cluster_centers_ for k in KM)

D_k = (sci_distance.cdist(cdata, cent, 'euclidean') for cent in centroids)
dist = (np.min(D, axis=1) for D in D_k)
avgWithinSS = [sum(d) / cdata.shape[0] for d in dist]
plt.plot(K, avgWithinSS, 'b*-')
plt.grid(True)
plt.xlabel('Количество кластеров')
plt.ylabel('Средняя сумма квадратов в кластере')
plt.show()
```

Рисунок 1.19 – Реализация в Python метода локтя для определения количества кластеров

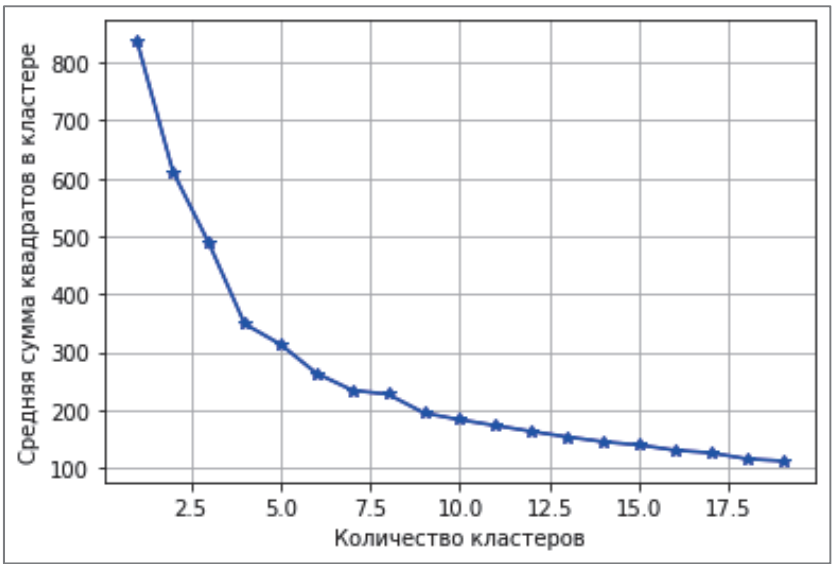


Рисунок 1.20 Метод локтя для определения количества кластеров

По рисунку 1.20 видно, что оптимальным является значение $k = 5$. При таком значении k регионы будут группироваться по пяти кластерам.

Далее реализуем кластерный анализ методом K-средних. В скрипте Python будем использовать функцию KMeans из пакета sklearn (Рисунок 1.21).

```
#Кластерный анализ методом k-средних
n_clusters = 5
means_cluster = sk_cluster.KMeans(n_clusters=n_clusters, random_state=111)
columns = ["X11", "X12", "X13", "X14", "X15", "X16", "X17", "X21", "X22","X23","X31",
           "X32","X41","X42"]
est = means_cluster.fit(data[columns])
clusters = est.labels_
data['cluster'] = clusters

#Выведем информацию о кластерах
for c in range(n_clusters):
    cluster_members=data[data['cluster'] == c][:]
    print('Cluster{}(n={}):'.format(c, len(cluster_members)))
    #print('-'* 17)
print(data.groupby(['cluster']).mean())
```

Рисунок 1.21 Реализация кластерного анализа методом k-средних в Python

После выполнения кластеризации по методу K-средних проведем анализ полученных результатов (Рисунок 1.22).

```
In [14]: runfile('E:/ВКР Н/Кластерный анализ/Cluster_K.py', wdir='E:/ВКР Н/Кластерный анализ')
Cluster0(n=34):
Cluster1(n=24):
Cluster2(n=12):
Cluster3(n=5):
Cluster4(n=3):
             X11       X12        X13  ...         X32       X41       X42
cluster                                 ...
0       -0.132206 -0.012029  -4.855912  ...  106.523529  0.009197  0.017994
1       -0.037583  0.001833  -0.883333  ...  169.408333  0.009774  0.017189
2       -2.383500 -0.017833 -24.062083  ...   35.266667  0.008477  0.019773
3        0.134600  0.001800   0.556800  ...  116.980000  0.007478  0.017682
4       -7.091000 -0.033000  -3.979667  ...  685.400000  0.007770  0.017644

[5 rows x 14 columns]

In [15]:
```

Рисунок 1.22 Результат кластерного анализа в консоли Python

Мы получили следующее распределение регионов по кластерам: нулевой кластер – 34 региона, первый кластер – 24 региона, второй кластер – 12 регионов, третий кластер – 5 регионов и четвертый кластер – 3 региона.

Для выгрузки средних значений по кластерам необходимо использовать библиотеки pandas и tkinter. Код выгрузки данных из Python будет выглядеть следующим образом (Рисунок 1.23).

```
40
41  import pandas as pd
42  import tkinter as tk
43  from tkinter import filedialog
44
45
46  a = data.groupby(['cluster']).mean()
47
48  df = pd.DataFrame(a)
49
50
51  root= tk.Tk()
52
53  c1 = tk.Canvas(root, width = 300, height = 300, bg = 'lightsteelblue2', relief
54  c1.pack()
55
56  def exportExcel ():
57      global df
58
59      export_file_path = filedialog.asksaveasfilename(defaultextension='.xlsx')
60      df.to_excel (export_file_path, index = False, header=True)
61
62  saveAsButtonExcel = tk.Button(text='Export Excel', command=exportExcel, bg='gre
63  c1.create_window(150, 150, window=saveAsButtonExcel)
64
65  root.mainloop()
```

Рисунок 1.23 Выгрузки результатов кластерного анализа из Python в Excel

Таким образом, мы получили средние значения показателей по каждому кластеру. Для определения ключевых факторов, определивших различия по кластерам, может быть проведен корреляционно-регрессионный анализ с использованием соответствующих библиотек языка программирования Python.

Самостоятельная работа

1. Используя предложенный файл “Clusters”, нормализируйте данные.

2. На основе ранее нормализованных данных реализуйте графическое представление иерархической кластеризации (дендограмму).

Вопросы для самоконтроля

1. Какие задачи решает кластерный анализ?
2. Какие требования к данным представляет кластерный анализ?
3. В чем состоит суть метода k-средних?
4. Для чего применяется метод локтя в кластерном анализе?
5. Какие библиотеки Python и для чего применялись в реализации кластерного анализа?

Раздел 2
Моделирование больших данных

Лабораторная работа № 2.1
«Построение модели линейной регрессии»

Теоретические положения

При наличии очищенных данных и хорошем понимании характера изучаемого набора данных, необходимо переходить к следующему этапу – моделирования данных.

Здесь как правило применяются методы машинного обучения, обработки/анализа данных, статистики. Как и весь процесс анализа больших данных, этап моделирования является итерационным процессом. Процесс построения модели состоит из следующих шагов:

- Выбор метода моделирования и переменных для включения в модель.
- Реализация модели.
- Диагностика качества и сравнение моделей.

Выбор правильной модели здесь – ваша основная задача и обязанность. Необходимо учесть качество модели и соответствие модели целям исследования.

После того как модель будет выбрана, ее необходимо реализовать в программном коде. И здесь нам потребуются библиотеки Python, о которых мы говорили ранее, например, StatModels, Scikit-learn и другие. Эти библиотеки поддерживают многие популярные методы моделирования. Программирование подобных моделей является делом нетривиальным и наличие таких библиотек существенно облегчает процесс.

Как видно из следующего кода, построить модели регрессии с использованием библиотек достаточно просто:

```
#Загрузим csv-файл с данными для анализа
from pandas import read_csv
data = read_csv('Analysis.csv',';')
data.head()
#Построение модели регрессии
import statsmodels.api as sm
#определим переменные x и y
x1 = data[['X1', 'X4', 'X5','X7', 'X8']]
y1 = data['Y2']
x1 = sm.add_constant(x1)
# Построим модель на основе метода наименьших квадратов
model = sm.OLS(y1, x1).fit()
predictions = model.predict(x1)
# Сформируем вывод
model.summary()
```

```
print(model.summary())
```

Как видно, построение множественной модели регрессии заняло всего 11 строк кода. Символом «#» обозначены текстовые комментарии кода.

Полученные результаты интерпретируются классическим образом: качество модели (R-квадрат), коэффициенты регрессии и их достоверность (p-value).

Принцип диагностики качества модели простой: модель должна работать для незнакомых данных. Существует множество разных метрик погрешности, например, среднеквадратическая ошибка (RMSE), которая в Python реализуется достаточно просто:

```
import math
y_actual = [1,2,3,4,5]
y_predicted = [1.6,2.5,2.9,3,4.1]

MSE = np.square(np.subtract(y_actual,y_predicted)).mean()

RMSE = math.sqrt(MSE)
print("Root Mean Square Error:\n")
print(RMSE)
```

Метрика RMSE очень проста: для каждого прогноза проверяется его отклонение от истинного значения, отклонение возводится в квадрат, а погрешности всех прогнозов суммируются и делятся на количество прогнозных значений. Далее, из полученной величины, извлекается корень. После получения рабочей модели все готово для перехода к следующей стадии.

Задание:

Условие. Имеются данные об эффективности государственной поддержки сельского хозяйства регионов Российской Федерации и эффективности сельскохозяйственного производства – рентабельности субсидий (Y1), производства (Y2) и средней урожайности зерновых в регионе (Y3). Показатели представлены в формате Хномер показателя. Показатели X1-X8 – факторные (объясняющие) переменные.

Требуется загрузить файл «Analysis» с данными в среду Spyder или Google Colab, построить модель множественной регрессии. Сделать выводы.

Данные: https://disk.yandex.ru/d/MwT4j-RH1v7OMA

Методические указания к выполнению лабораторной работы

Загрузим данные в среду Spyder:

Рисунок 2.1 Загрузка данных в среду разработки Spyder

Следующим этапом построим матрицу парных коэффициентов корреляции, чтобы проверить взаимосвязь между всеми представленными переменными [11]. Для этого импортируем библиотеки Seaborn и Numpy. Для наглядности, используя функцию sns.heatmap() и построим матрицу парных коэффициентов корреляции в виде тепловой карты [13].

Реализация будет выглядеть следующим образом (Рисунок 2.2):

```
#Построим матрицу корреляции в виде тепловой карты
import seaborn as sns
import numpy as np
matrix = np.triu(data.corr())
sns.heatmap(data.corr(), annot=True, mask=matrix, fmt='.1g')
```

Рисунок 2.2 Реализация функции тепловой карты

В итоге получим следующую матрицу парных коэффициентов корреляции:

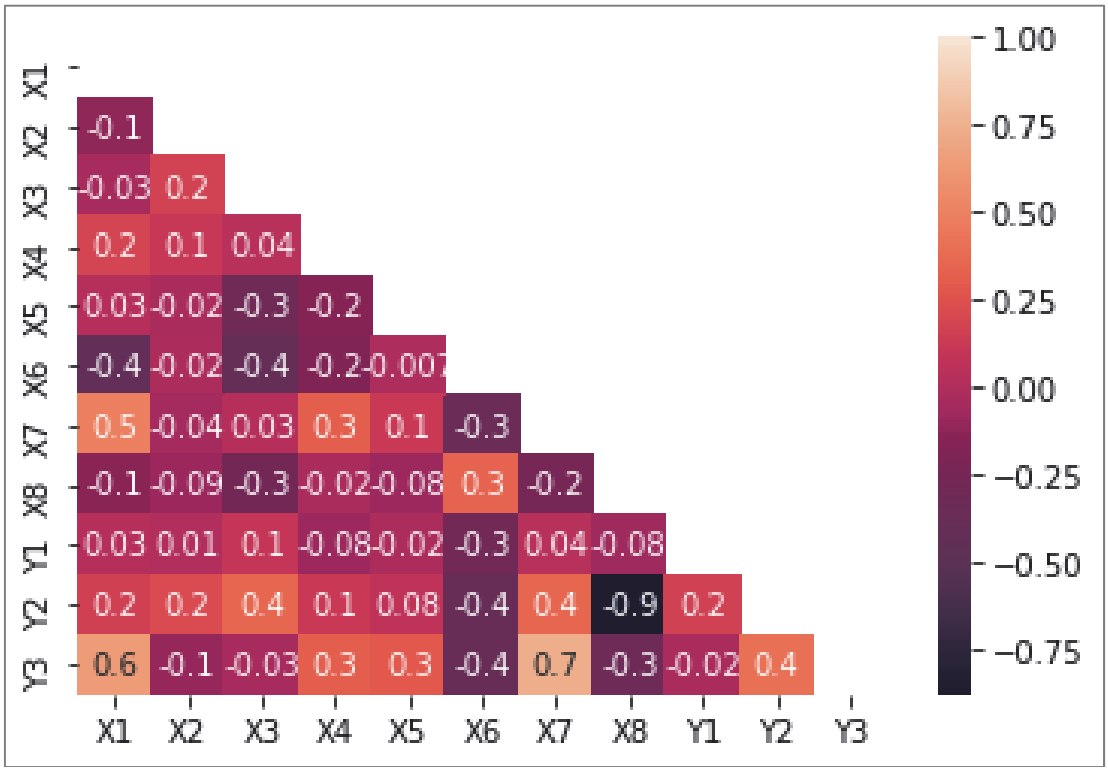


Рисунок 2.3 Матрица парных коэффициентов корреляции

Анализируя матрицу (Рисунок 2.3), отметим, что коррелируют между собой следующие пары переменных (коэффициент больше или равен значения 0,3):

Y1 – X6;

Y2 – X3, X6, X7, X8

Y3 – X1, X4, X5, X6, X7, X8

Также анализ матрицы парных коэффициентов регрессии показал, что на результативную переменную Y1 практически все переменные оказывают

влияние ниже среднего, только переменная X6 оказывает средний уровень влияния на результативную переменную Y1. Поэтому далее построим модель регрессии между этими переменными.

На первом этапе определим переменные и построим диаграмму рассеивания (Рисунок 2.4 и 2.5).

```
6
7   #определим переменные x и y
8   x = data['X6']
9   y = data['Y1']
10
11  #импортируем библиотеку для построения диаграммы рассеивания
12  import matplotlib.pyplot as plt
13
14  #построим диаграмму рассеивания
15  plt.plot(x, y, 'o', color = 'green')
16  plt.show()
```

Рисунок 2.4 Реализация построения диаграммы рассеивания

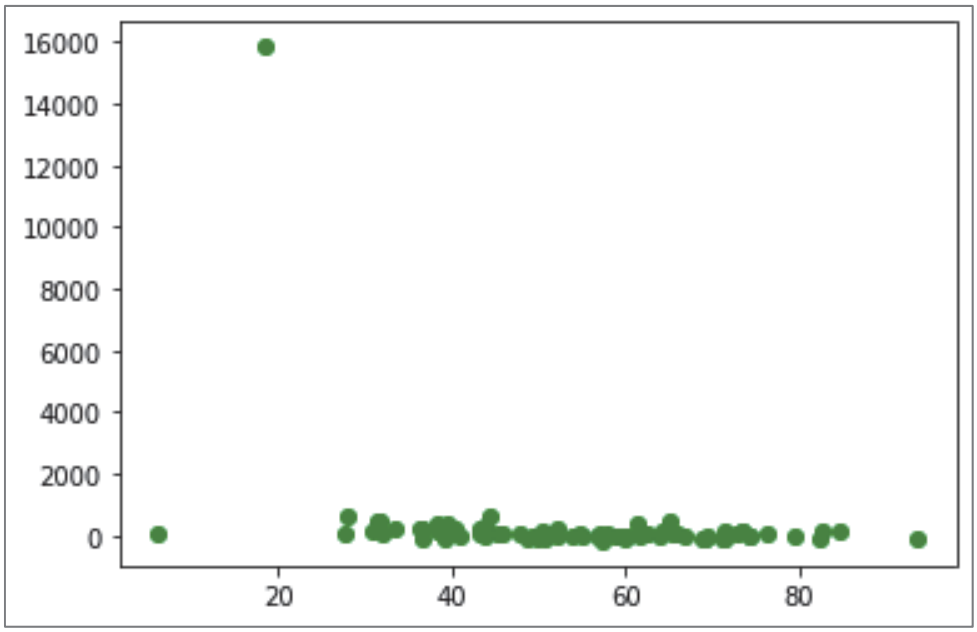


Рисунок 2.5 Диаграмма рассеивания (зависимости)

Мы видим, что имеются выпадающие значения, которые не позволят проводить дальнейший анализ корректно. Эти значения необходимо удалить. Диаграмма рассеивания примет вид (Рисунок 2.6).

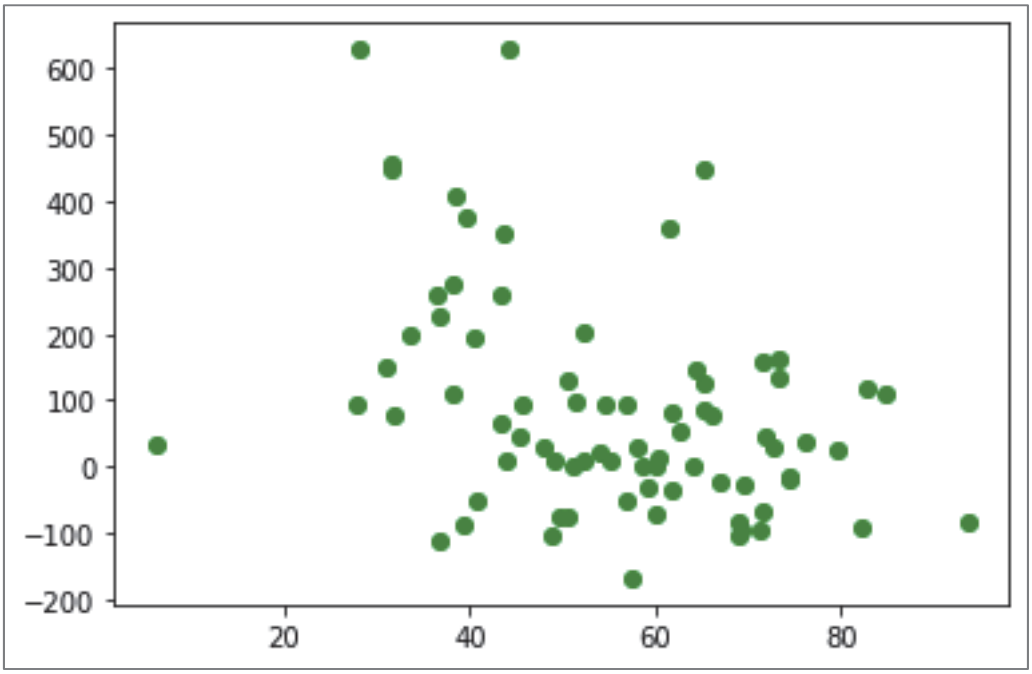


Рисунок 2.6 Диаграмма рассеивания (зависимости) после удаления выброса

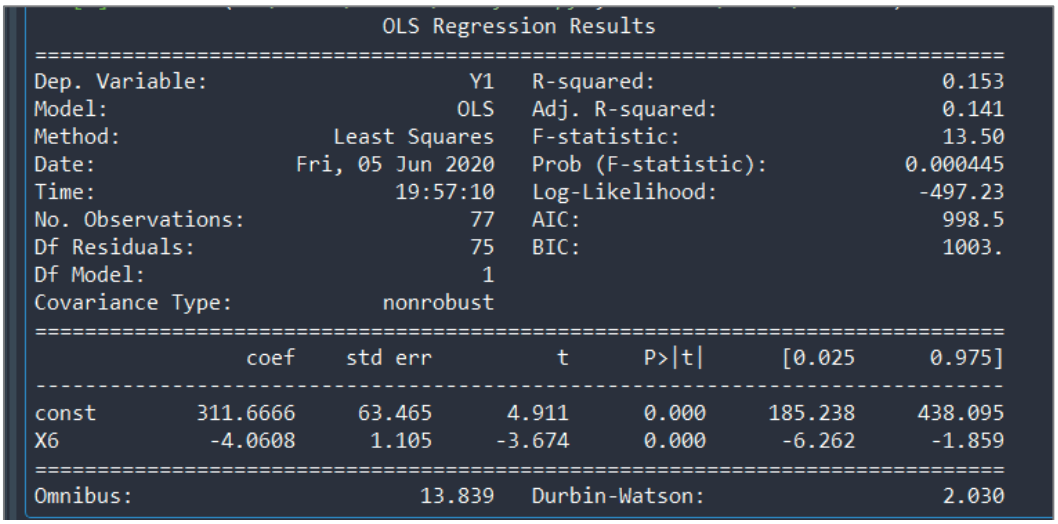
Теснота связи находится на среднем уровне и подтверждается обратная зависимость между переменными.

Далее выведем регрессионную модель Y1 = a + bX6 и другие статистические параметры корреляционно-регрессионного анализа (Рисунок 2.7) [6].

```
13  #Построение модели регрессии Y1 и X6
14  import statsmodels.api as sm
15
16  #определим переменные x и y
17  x = data['X6']
18  y = data['Y1']
19  #Добавим в модель условное начало
20  x = sm.add_constant(x)
21
22  # Построим модель на основе метода наименьших квадратов
23  model = sm.OLS(y, x).fit()
24  predictions = model.predict(x)
25
26  #Сформируем вывод
27  model.summary()
28  print(model.summary())
```

Рисунок 2.7 Реализация модели парной линейной регрессии методом наименьших квадратов

Вывод итогов, содержаний модель регрессии, коэффициент детерминации, F- и t-тесты и другие статистические характеристики выглядит следующим образом (Рисунок 2.8).



OLS Regression Results

Dep. Variable:	Y1	R-squared:	0.153
Model:	OLS	Adj. R-squared:	0.141
Method:	Least Squares	F-statistic:	13.50
Date:	Fri, 05 Jun 2020	Prob (F-statistic):	0.000445
Time:	19:57:10	Log-Likelihood:	-497.23
No. Observations:	77	AIC:	998.5
Df Residuals:	75	BIC:	1003.
Df Model:	1		
Covariance Type:	nonrobust		

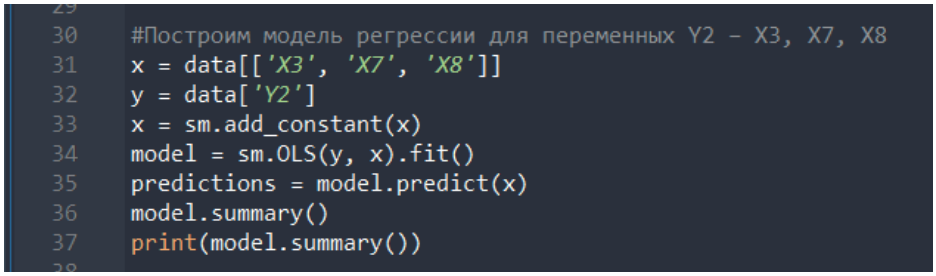
	coef	std err	t	P>|t|	[0.025	0.975]
const	311.6666	63.465	4.911	0.000	185.238	438.095
X6	-4.0608	1.105	-3.674	0.000	-6.262	-1.859

Omnibus:	13.839	Durbin-Watson:	2.030

Рисунок 2.8 Результаты регрессионного анализа для переменных Y1 и X6

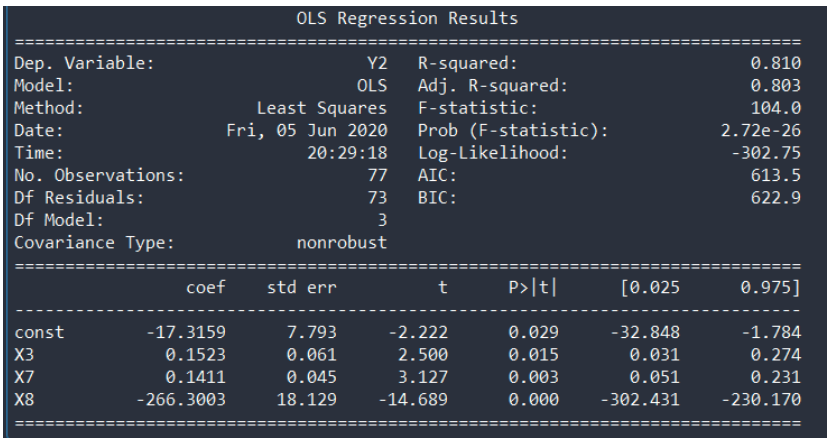
Построенная модель регрессии подтвердила обратную зависимость между переменными Y1 и X6. При увеличении переменной X6 на 1%, величина показателя Y1 сократиться на 4.1%.

Построение регрессии для переменной Y2. Стоит отметить, что при построении модели регрессии, значимость переменной X6 составила 66,4%, что значительно превышает допустимый уровень 5%. Поэтому указанная переменная не включена в дальнейшие этапы построения регрессии (Рисунок 2.9 и 2.10).



```
30  #Построим модель регрессии для переменных Y2 – X3, X7, X8
31  x = data[['X3', 'X7', 'X8']]
32  y = data['Y2']
33  x = sm.add_constant(x)
34  model = sm.OLS(y, x).fit()
35  predictions = model.predict(x)
36  model.summary()
37  print(model.summary())
```

Рисунок 2.9 Реализация модели регрессии для переменных Y2 – X3, X7, X8



OLS Regression Results

Dep. Variable:	Y2	R-squared:	0.810
Model:	OLS	Adj. R-squared:	0.803
Method:	Least Squares	F-statistic:	104.0
Date:	Fri, 05 Jun 2020	Prob (F-statistic):	2.72e-26
Time:	20:29:18	Log-Likelihood:	-302.75
No. Observations:	77	AIC:	613.5
Df Residuals:	73	BIC:	622.9
Df Model:	3		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-17.3159	7.793	-2.222	0.029	-32.848	-1.784
X3	0.1523	0.061	2.500	0.015	0.031	0.274
X7	0.1411	0.045	3.127	0.003	0.051	0.231
X8	-266.3003	18.129	-14.689	0.000	-302.431	-230.170

Рисунок 2.10 Результаты регрессионного анализа для переменных Y2 – X3, X7, X8

В прямой зависимости с переменной Y2 находятся такие факторные показатели как X3, X7. Показатель X8 находится в обратной зависимости к переменной Y2.

Построение регрессии для переменной Y3 (Рисунок 2.11 и 2.12).

```
#Построим модель регрессии для переменных Y3 – X1, X4, X5, X7, X8
x = data[['X1', 'X4', 'X5', 'X7', 'X8']]
y = data['Y3']
x = sm.add_constant(x)
model = sm.OLS(y, x).fit()
predictions = model.predict(x)
model.summary()
print(model.summary())
```

Рисунок 2.11 Реализация модели регрессии для переменных Y3 – X1, X4, X5, X7, X8

OLS Regression Results

Dep. Variable:	Y3	R-squared:	0.744
Model:	OLS	Adj. R-squared:	0.726
Method:	Least Squares	F-statistic:	41.22
Date:	Fri, 05 Jun 2020	Prob (F-statistic):	1.11e-19
Time:	20:37:02	Log-Likelihood:	-231.36
No. Observations:	77	AIC:	474.7
Df Residuals:	71	BIC:	488.8
Df Model:	5		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	-0.7171	2.379	-0.301	0.764	-5.461	4.026
X1	0.0739	0.014	5.335	0.000	0.046	0.102
X4	1.1623	0.568	2.048	0.044	0.030	2.294
X5	0.0276	0.007	3.742	0.000	0.013	0.042
X7	0.1354	0.022	6.223	0.000	0.092	0.179
X8	-15.8981	7.024	-2.263	0.027	-29.904	-1.892

Рисунок 2.12 Результаты регрессионного анализа для переменных Y3 – X1, X4, X5, X7, X8

На показатель Y3, прямое воздействие оказывают такие факторы X1, X4, X5, X7. Отрицательное влияние оказывает показатель X8.

Самостоятельная работа

1. Используя предложенный файл “Analysis”, постройте модель регрессии методом шаговой регрессии.
2. Используя предложенный файл “Analysis”, постройте модель регрессии методами Ridge и Lasso регрессии.
3. Оцените качество построенных моделей (в том числе методом OLS) на основе MSE и RSME критериев.

Вопросы для самоконтроля

1. Какие методы оценки параметров лежат в основе построения линейных моделей регрессии?

2. Перечислите показатели, используемые для оценки качества моделей регрессии.
3. В чем состоит назначение матрицы парных коэффициентов корреляции?
4. Дайте интерпретацию максимальному количеству показателей результатов построения регрессии (Рисунок 2.12).
5. Все ли параметры построенных моделей статистически достоверны?

Лабораторная работа № 2.2
«Построение модели probit-регрессии»

Теоретические положения

Пробит-модель – тип регрессионных моделей, которые обычно используются в статистическом анализе, особенно в области бинарной классификации. Это означает, что интересующий результат может принимать только два возможных значения / класса. В большинстве случаев эти модели используются для прогнозирования того, произойдет ли что-то в виде бинарного результата. Например, банк может захотеть узнать, может ли конкретный заемщик не выплатить кредит или нет [5].

Пробит-модель – это разновидность статистической модели, которая используется для прогнозирования вероятности наступления события. Эти модели также называются моделями пробит-регрессии. Пробит-модель похожа на логит-модель, но основана на пробит-функции, а не на логистической функции. Пробит-функция также называется функцией пробит-связи. В пробит-модели кумулятивная функция распределения $\Phi(\theta)$ стандартного нормального распределения используется для моделирования взаимосвязи между предикторами и вероятностью наступления события (Рисунок 2.13). Выходные данные пробит-модели также варьируются от 0 до 1, как и у логит-модели.

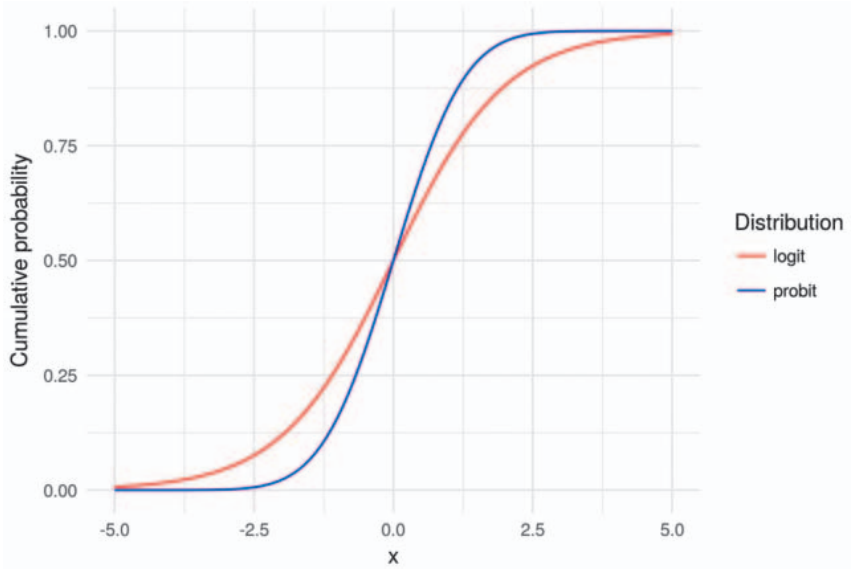


Рисунок 2.13 Графическое представление Logit и Probit моделей

Формула пробит-модели выглядит следующим образом:

$Pr(Y = 1|X) = \Phi(Z)$ (2.1)

где, $Z = b0 + b1X1 + b2X2 + ... + bnXn$ (2.2)

где Y - зависимая переменная, представляющая вероятность того, что событие произойдет (следовательно, Y = 1) при заданных переменных X.

Z - линейная комбинация независимых переменных (X) с коэффициентами (b0, b1, b2...bn). В случае логит-модели мы используем логистическую или сигмовидную функцию вместо Ф, которая является кумулятивной функцией стандартного нормального распределения. Параметры (такие как b0, b1 и т.д.) оцениваются с использованием метода оценки максимального правдоподобия. Это также справедливо для логит-моделей. Вы можете заметить, что ключевым отличием между логит- и пробит-моделями является сигмовидная или логистическая функция и кумулятивная функция нормального распределения соответственно [5].

Задание:

Условие. Имеются данные о финансовом состоянии сельскохозяйственных организаций (файл «FinCond»). Переменная Y – является фиктивной и обозначает «да» - 1, организация является банкротом и «нет» - 0, не является банкротом. Переменные X1 и X2 характеризуют финансовое состояние организаций.

Требуется загрузить данные в Spyder, рассчитать описательную статистику (Descriptive statistics), построить probit-модель, позволяющую прогнозировать банкротство организации, определить вероятность банкротства «средней» организации и предельный эффект факторов, включенных в модель. Сделать выводы.

Данные: https://disk.yandex.ru/i/MCRoljXpm3Q49w

Методические указания к выполнению лабораторной работы

1. Необходимо загрузить данные в память с помощью пакета «pandas». Данные будут считываться как «фрейм» данных pandas. Также загрузим библиотеку «numpy» для манипуляций с массивом, и будем использовать пакет «statsmodel» для регрессии Probit. Загрузим все требуемые библиотеки.

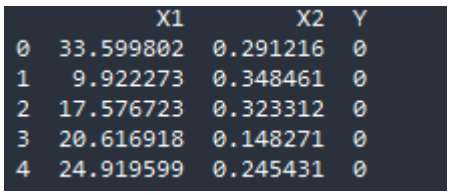
```
import pandas as pd
import numpy as np
from statsmodels.discrete.discrete_model import Probit
import statsmodels.api as sm
import seaborn as sns
```

2. Загрузим данные в Spyder.

```
data = pd.read_excel('FinCond.xlsx')
```

Напечатаем первые несколько строк данных, с помощью функции head.

```
print(data.head())
```

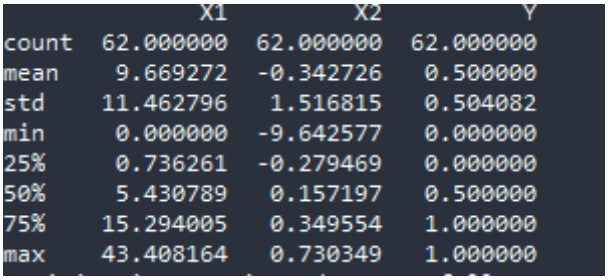


	X1	X2	Y
0	33.599802	0.291216	0
1	9.922273	0.348461	0
2	17.576723	0.323312	0
3	20.616918	0.148271	0
4	24.919599	0.245431	0

Рисунок 2.14 Результат загрузки исходных данных в Spyder

3. Рассчитаем описательную статистику с применением функции «describe()». Это позволит провести предварительное исследование данных.

print(data.describe())



	X1	X2	Y
count	62.000000	62.000000	62.000000
mean	9.669272	-0.342726	0.500000
std	11.462796	1.516815	0.504082
min	0.000000	-9.642577	0.000000
25%	0.736261	-0.279469	0.000000
50%	5.430789	0.157197	0.500000
75%	15.294005	0.349554	1.000000
max	43.408164	0.730349	1.000000

Рисунок 2.15 Расчет описательной статистики

Всего в совокупности 62 организации, количество организаций по финансовому состоянию разделилось ровно пополам.

4. Выберем переменные для построения пробит-регрессии.

Y = data["Y"]

X = data[["X1", "X2"]]

5. Пропишем «условное начало» в модели:

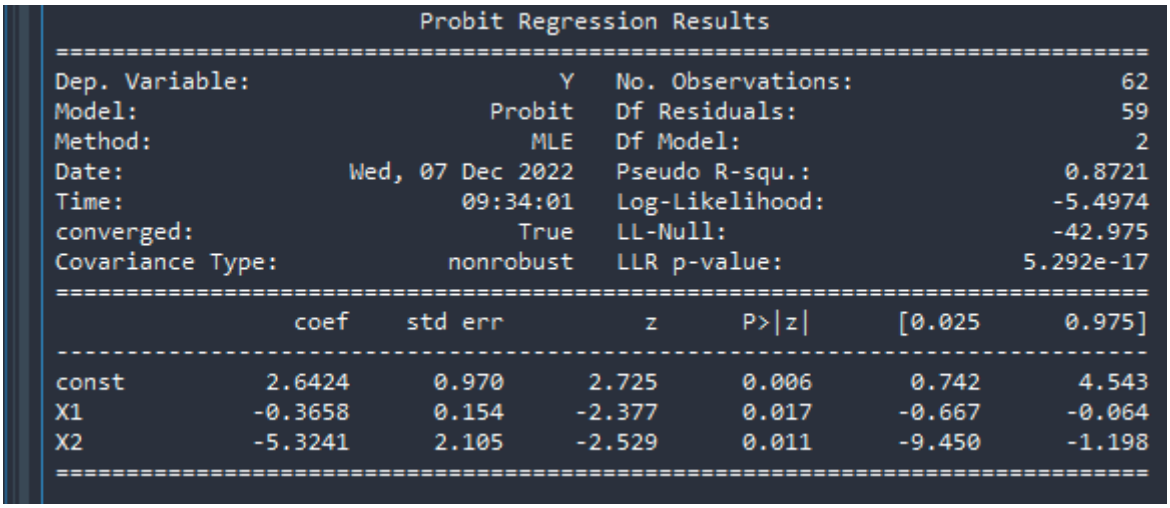
X = sm.add_constant(X)

6. Применим метод Probit к переменным, построим модель и выведем результат вычислений:

model = Probit(Y, X.astype(float))

probit_model = model.fit()

print(probit_model.summary())



Probit Regression Results

Dep. Variable:	Y	No. Observations:	62
Model:	Probit	Df Residuals:	59
Method:	MLE	Df Model:	2
Date:	Wed, 07 Dec 2022	Pseudo R-squ.:	0.8721
Time:	09:34:01	Log-Likelihood:	-5.4974
converged:	True	LL-Null:	-42.975
Covariance Type:	nonrobust	LLR p-value:	5.292e-17

	coef	std err	z	P>|z|	[0.025	0.975]
const	2.6424	0.970	2.725	0.006	0.742	4.543
X1	-0.3658	0.154	-2.377	0.017	-0.667	-0.064
X2	-5.3241	2.105	-2.529	0.011	-9.450	-1.198

Рисунок 2.16 Результаты построения пробит-модели

Модель и параметры модели статистически достоверны на уровне существенности 5%. Псевдо-коэффициент детерминации близок к 1, что так же положительно характеризует построенную модель.

Полученные коэффициенты (столбец «coef»), как известно, не интерпретируются, но могут быть использованы для определения вероятности наступления того или иного события, а также для расчета предельных эффектов каждой переменной.

7. Определим вероятность банкротства одной «средней» организации. Средние значения

Рассчитаем среднее значение Z:

$\overline{Z}$ = b0 + b1$\overline{X1}$ + b2$\overline{X2}$

Можно записать выражение в среде Python

z = 2.6424-0.3658*9.669272-5.3241*(-0.342726)

Далее представим прогноз о вероятности ее банкротства, подставив значение z в формулу стандартного нормального распределения:

$$\frac{1}{\sqrt{2\pi}} e^{-\frac{z^2}{2}} \quad (2.3)$$

Пропишем выражение в среде разработки:

```
pb=(1/(math.sqrt(2*3.14)))*(2.72)**(-0.5*(z*z))

print(pb)
```

```
0.93
0.25885429182896547
```

Рисунок 2.17 Результаты прогноза вероятности наступления события

Первое значение – величина z, второе – вероятность наступления события. Исходя из полученного результата, можно сделать вывод, что вероятность банкротства «Средней» организации равна 26%.

8. Определим предельный эффект от переменной X1. Для расчета предельного эффекта вероятность (pb) умножим на величину коэффициента при X1 – b1:

```
px1=(1/(math.sqrt(2*3.14)))*(2.72)**(-0.5*(z*z))*(-0.3658)

print(round(px1,4))
```

```
-0.0947
```

Рисунок 2.18 Расчет предельного эффекта переменной X1

Таким образом, увеличение значение переменной X1 на 1 единицу (в соответствующих единицах измерения) приведет к сокращению вероятности наступления банкротства на 9,5%.

Самостоятельная работа

1. Используя предложенный файл “FinCond”, постройте модель логит-регрессии.
2. Рассчитайте параметры качества моделей пробит- и логит-регрессии.

Вопросы для самоконтроля

1. В решении каких задач применяются модели пробит- и логит-регрессии?
2. Перечислите и укажите назначение используемых в работе библиотек Python.
3. Как определить предельный эффект переменной?

Лабораторная работа № 2.3
«Сравнение моделей машинного обучения на основе Python библиотеки lazypredict»

Теоретические положения

LazyPredict – это библиотека для Python, которая помогает быстро строить модели машинного обучения. Она позволяет строить и сравнивать несколько моделей на наборах данных [9].

К особенностям LazyPredict относится:

– Поддержка широкого спектра алгоритмов машинного обучения. Среди них: линейная регрессия, деревья решений, случайные леса, градиентный boosting, нейронные сети и другие.

– Автоматический выбор модели. LazyPredict автоматически выбирает лучшую модель для заданного набора данных на основе входных признаков и целевой функции вывода.

– Оптимизация гиперпараметров. LazyPredict автоматически оптимизирует гиперпараметры для каждой модели для улучшения производительности.

Поддержка регрессии и классификации. LazyPredict автоматически определяет тип проблемы на основе целевой функции вывода.

Библиотеки Python, необходимые для работы с LazyPredict: pandas, numpy, scipy, xgboost, lightgbm, catboost и TensorFlow [9].

Задание:

Условие. Имеются данные о ценах на объекты недвижимости в сельской местности (файл «Rural_prices»).

Требуется загрузить данные в Spyder, рассчитать описательную статистику (Descriptive statistics), используя библиотеку lazypredict

определить лучшую модель регрессии для прогнозирования цен на объекты недвижимости. Сделать выводы.

Данные: https://disk.yandex.ru/d/RMwt-T5lH9yqoQ

Методические указания к выполнению лабораторной работы

Для установки Lazy Predict, необходимо запустить команду в терминале:

pip install lazypredict

После установки библиотеки импортируем все необходимые для работы с lazypredict библиотеки.

```
#импортируем необходимые библиотеки
import pandas as pd
import seaborn as sns
from sklearn.model_selection import train_test_split
import lazypredict
from lazypredict.Supervised import LazyRegressor
```

Рисунок 2.18 Импорт требуемых библиотек

Далее загрузим данные в Spyder и выберем переменные для построения модели.

```
#загрузим данные
data = pd.read_csv('Rural_prices.csv', sep=";", decimal=",")

#выберем переменные для модели
x1 = data.drop('sale price', axis=1)
y1 = data['sale price']
```

Рисунок 2.19 Импорт данные и определение переменных модели

Атрибуты sep=";", decimal="," необходимы для того, чтобы данные в csv-файле представлялись в табличном виде и были пригодны для дальнейшего анализа без ограничений.

	sale price	lot size	#bedroom	#bath	...	gas	air cond	#garage	desire loc
0	42000	5850	3	1	...	0	0	1	0
1	38500	4000	2	1	...	0	0	0	0
2	49500	3060	3	1	...	0	0	0	0
3	60500	6650	3	1	...	0	0	0	0
4	61000	6360	2	1	...	0	0	0	0
..	...	...	...	...	...	...	...	...	...
541	91500	4800	3	2	...	0	1	0	0
542	94000	6000	3	2	...	0	1	0	0
543	103000	6000	3	2	...	0	1	1	0
544	105000	6000	3	2	...	0	1	1	0
545	105000	6000	3	1	...	0	1	1	0

[546 rows x 12 columns]

Рисунок 2.20 Датасет для построения моделей

Мы видим, что датасет представлен 546 наблюдениями (то есть объектами недвижимости) и 12 признаками, включая целевую переменную sale price, которую необходимо предсказать. К сожалению, мы не видим все переменных, по причине размеров выводимой в консоли таблицы. Для вывода названия всех столбцов воспользуемся возможностями библиотеки Pandas:

```
column_names = data.columns.tolist()
print(column_names)
```

```
['sale price',
 'lot size',
 '#bedroom',
 '#bath',
 '#stories',
 'driveway',
 'rec room',
 'basement',
 'gas',
 'air cond',
 '#garage',
 'desire loc']
```

Рисунок 2.20 Список переменных, участвующих в построении моделей регрессии

В результате мы получили полный список признаков, участвующих в построении и сравнении моделей регрессии.

'sale price' – цена продажи объекта недвижимости
'lot size' – размер объекта недвижимости
'#bedroom' – наличие спальни
'#bath' – наличие ванной комнаты
'#stories' – этажность
'driveway' – подъездная дорога
'rec room' – наличие комнаты отдыха
'basement' – наличие подвала
'gas' – наличие подключения дома к газу
'air cond' – наличие кондиционера
'#garage' – наличие гаража
'desire loc' – подходящее месторасположение дома

Для характеристики и изучения используемых данных рассчитаем описательную статистику, используя метод .describe().

	sale price	lot size	#bedroom	...	air cond	#garage	desire loc
count	546.00	546.00	546.00	...	546.00	546.00	546.00
mean	68121.60	5150.27	2.97	...	0.32	0.69	0.23
std	26702.67	2168.16	0.74	...	0.47	0.86	0.42
min	25000.00	1650.00	1.00	...	0.00	0.00	0.00
25%	49125.00	3600.00	2.00	...	0.00	0.00	0.00
50%	62000.00	4600.00	3.00	...	0.00	0.00	0.00
75%	82000.00	6360.00	3.00	...	1.00	1.00	0.00
max	190000.00	16200.00	6.00	...	1.00	3.00	1.00

Рисунок 2.21 Описательная статистика для датасета

Описательная статистика крайне полезна для настройки предельных величин ошибки RMSE, величина которой зависит от масштаба анализируемой переменной.

Далее разобьём представленную выборочную совокупность на тестовую и тренировочную выборки в соотношении 80 к 20.

```
# Установим разбиение выборки в соотношении 80/20
X_train, X_test, Y_train, Y_test = train_test_split(x1, y1, test_size=0.2, random_state=None)
```

Рисунок 2.22 Разбиение совокупности на тестовую и тренировочную

Парметр random_state установим в режиме None. Объем тестовой выборки установим в размере 20% (также допустима величина 30%).

Далее построим lazyregressor, который позволит нам автоматически построить более 30 моделей регрессии, натренировать их, протестировать и сравнить эффективность.

```
# определим и построим lazyregressor
clf = LazyRegressor(verbose=0,ignore_warnings=True, custom_metric=None)
models_train,predictions_train = clf.fit(X_train, X_train, Y_train, Y_train)
models_test,predictions_test = clf.fit(X_train, X_test, Y_train, Y_test)
```

Рисунок 2.22 Построение lazyregressor

Стоит отметить, что данная библиотека может также строить и автоматически сравнивать модели классификации на основе метода LazyClassifier() [4].

Далее визуализируем показатели эффективности моделей, использую библиотеки seaborn и matplotlib.

```
#визуализируем показатели эффективности моделей
# столбчатая диаграмма значений коэффициента детерминации
import matplotlib.pyplot as plt
import seaborn as sns
```

Рисунок 2.23 Библиотеки для визуализации эффективности моделей

Всего визуализируем три показателя эффективности – коэффициент детерминации, RSME, время, за которое выполняется каждый из алгоритмов построения модели.

```
#коэффициент детерминации = [0 если i < 0 иначе i for i in train.iloc[:,0] ]

plt.figure(figsize=(5, 10))
sns.set_theme(style="whitegrid")
ax = sns.barplot(y=predictions_train.index, x="R-Squared", data=predictions_train)
ax.set(xlim=(0, 1))
```

Рисунок 2.24 Построение столбчатой диаграммы значений коэффициента детерминации

Отметим, что в случае некорректного определения показателя детерминации, его значение будет обозначено величиной 0.

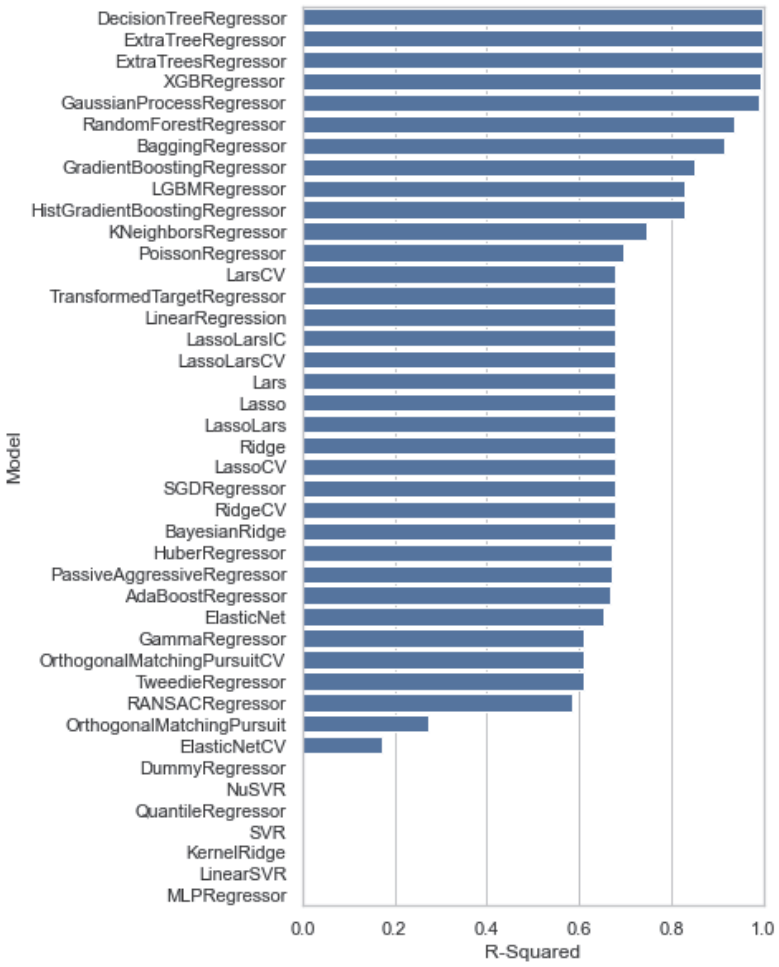


Рисунок 2.25 Столбчатая диаграмма значений коэффициента детерминации

Всего, как мы видим построено 35 моделей машинного обучения. В число самых эффективных по показателю R-квадрат вошли такие модели как DecisionTree, ExtraTreeRegressor, XGBRegressor.

Далее визуализируем метрику качества моделей машинного обучение RSME.

```
# столбчатая диаграмма значений RMSE
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(5, 10))
sns.set_theme(style="whitegrid")
ax = sns.barplot(y=predictions_train.index, x="RMSE", data=predictions_train)
ax.set(xlim=(0, 190000))
```

Рисунок 2.26 Построение столбчатой диаграммы значений RSME

Для отображения масштаба RSME нам пригодились знания, полученные при построении описательной статистики для данного датасета. При указании масштаба оси абсцисс, мы указали максимальное значение показателя “sale price”, что позволило нам корректно отразить величину RSME для данного датасета и соответствующего набора моделей.

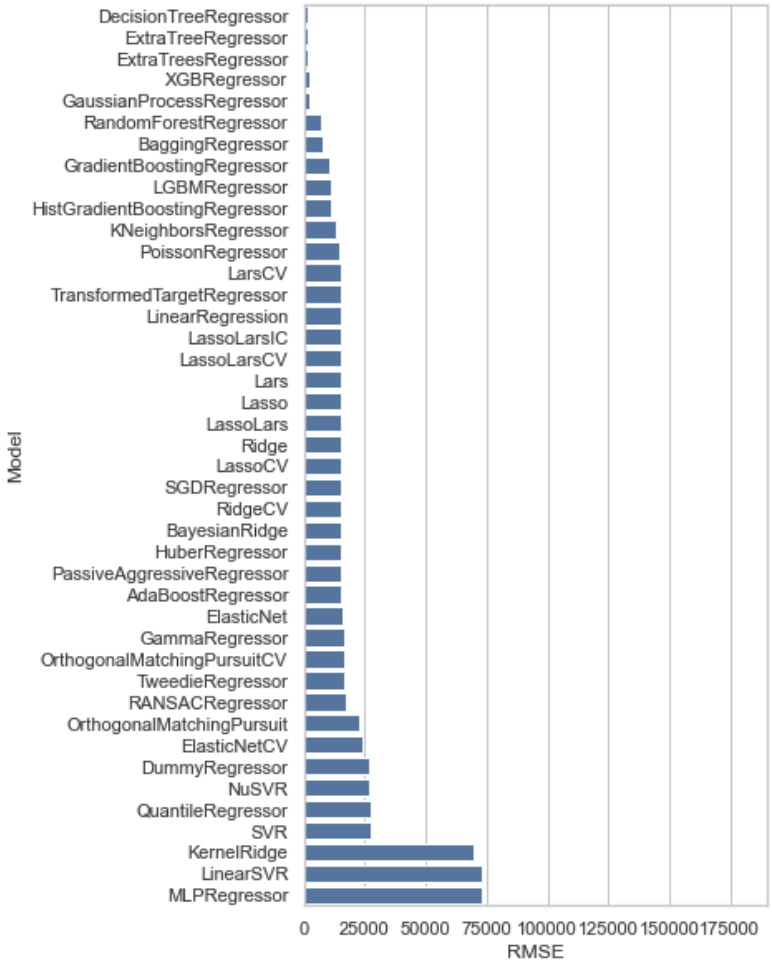


Рисунок 2.27 Столбчатая диаграмма значений RSME

Из графика видно, что максимальное значение RSME для данного датасета можно ограничить величиной 75000.

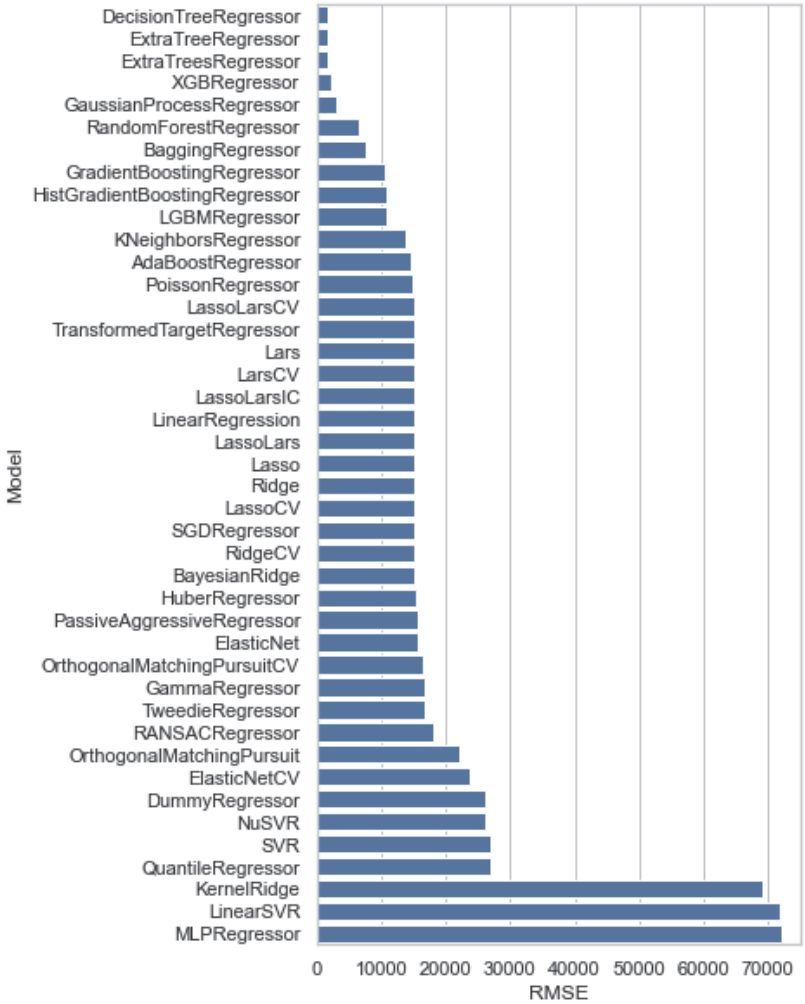


Рисунок 2.28 Столбчатая диаграмма значений RSME после ограничения максимальной величины

Наиболее эффективными, в данном случае показывающими наименьшую ошибку также являются модели DecisionTree, ExtraTreeRegressor, XGBRegressor.

И, наконец, визуализируем третий показатель качества моделей – время, за которое выполняется каждый из алгоритмов построения модели (time taken).

```
# столбчатая диаграмма значений времени выполнения алгоритма
import matplotlib.pyplot as plt
import seaborn as sns

plt.figure(figsize=(5, 10))
sns.set_theme(style="whitegrid")
ax = sns.barplot(y=predictions_train.index, x="Time Taken", data=predictions_train)
ax.set(xlim=(0, 0.5))
```

Рисунок 2.29 Построение столбчатой диаграммы значений времени выполнения алгоритма

Здесь мы также ограничим масштаб оси x коридором от 0 до 0.5 секунды, исходя из максимальных пределов затраченного времени.

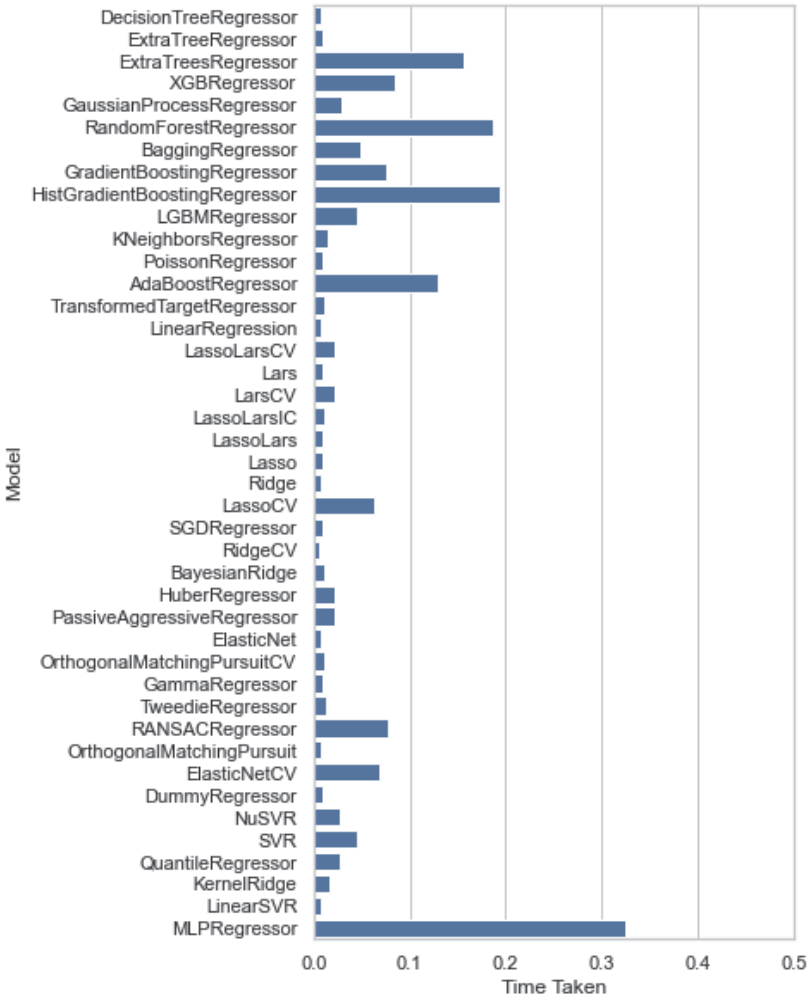


Рисунок 2.30 Столбчатая диаграмма значений времени выполнения алгоритма

По времени выполнения все алгоритмы отработали достаточно быстро. Наихудшим образом отработала модель MLPRegressor. В тоже время, чуть больше 0,3 секунды вполне приемлемый результат. Используя данную библиотеку далее при прогнозировании, можно использовать самые эффективные модели машинного обучения.

Самостоятельная работа

1. Используя данные из лабораторной работы, посвященной пробит-моделям, постройте модели классификации на основе метода LazyClassifier() и сравните их эффективность.
2. Визуализируйте результаты показателей эффективности моделей и выберите лучшую модель для задачи классификации.

Вопросы для самоконтроля

1. Для решения каких задач предназначена библиотека Lazy Predict?

2. По каким показателям сравнивается эффективность моделей при использовании библиотеки Lazy Predict?

3. Какие библиотеки используются для визуализации критериев эффективности моделей машинного обучения?

4. В каких пропорциях разбивается совокупность на тестовую и тренировочную выборки?

5. Для решения каких классов задач строит модели машинного обучения библиотека Lazy Predict?

Лабораторная работа № 2.4
«Выявление скрытых переменных на основе метода PCA»

Теоретические положения

Метод главных компонент (Principal Component Analysis, PCA) – один из самых популярных методов снижения размерности данных и выявления скрытых структур в наборах данных, что особенно актуально в контексте обработки большого объема данных при нехватке вычислительных мощностей и необходимости повышения эффективности вычислений. Метод получил широкое распространение, в том числе в сфере машинного обучения, статистики, анализа больших данных и так далее. Метод PCA позволяет преобразовать исходный набор данных так, чтобы новые переменные (которые носят имя «главных компонент») объяснили как можно больший объем вариации данных при минимальном числе переменных. С методом PCA связаны такие понятия как корреляция, ковариационная матрица, собственные значения и вектора.

Метод PCA основан на поиске линейных комбинаций исходных переменных, которые объясняют максимальный объем вариации данных. Линейные комбинации носят имя главных компонент. Главные компоненты от первой к последующим объясняют объем вариации по принципу от максимального до минимального объясненного объема вариации.

Реализация метода PCA потребует прохождения следующих шагов: нормализация данных, расчет матрицы ковариаций, поиск собственных значений и собственных векторов, выбор числа главных компонент, построение новых координат. Снижение размерности данных на основе метода PCA основано на объединении переменных, тесно коррелирующих друг с другом.

Нормализация данных осуществляется на формуле:

$$X'_{ij} = \frac{X_{ij} - \mu_j}{\sigma_j} \quad (2.4)$$

где μ_j и σ_j – среднее значение и стандартное отклонение j-го признака соответственно.

Важно помнить, что данный метод предполагает линейную зависимость между переменными, поэтому в случае нелинейный зависимостей данный метод может быть неэффективен.

В работе применяется библиотека Scikit-learn, потому что она уже содержит готовую реализацию PCA и позволяет строить графики каменистой осыпи.

Задание:

Условие. Имеются данные о ценах на объекты недвижимости в сельской местности (файл «Rural_prices»).

Требуется загрузить данные в Spyder, объяснить показателя «sale prices» с использованием разных характеристик объектов недвижимости. Сделать выводы.

Данные: https://disk.yandex.ru/d/RMwt-T5lH9yqoQ

Методические указания к выполнению лабораторной работы

Импортируем необходимые библиотеки, загрузим данные в среду Spyder и проведем их стандартизацию.

```
#импортируем необходимые библиотеки
import pandas as pd
import seaborn as sns
from sklearn import preprocessing
from sklearn.decomposition import PCA
import pylab as plt


#загрузим данные
data = pd.read_csv('Rural_prices.csv', sep=";", decimal=",")

#выберем переменные для модели
x1 = data.drop('sale price', axis=1)
y1 = data['sale price']

# Стандартизация данных
x1 = preprocessing.StandardScaler().fit(x1).transform(x1)
```

Рисунок 2.31 Загрузки библиотек, данных и их стандартизация для задач PCA

Далее проведем анализ главных компонент и визуализируем результаты.

```
# Создание экземпляра класса анализа главных компонент
model = PCA()

# Применение PCA к свободным переменным для поиска возможности свертки их в меньшее количество переменных
# Результат преобразуется в массив для использования вновь созданных данных
results = model.fit(x1)
Z = results.transform(x1)

# График объяснимой дисперсии переменных: в данном случае график каменистой осыпи
plt.plot(results.explained_variance_)
# Отображение графика
plt.show()
```

Рисунок 2.32 Построение и визуализация модели PCA

В результате преобразования данных, их свертки методом PCA получаем график объяснения вариации признака факторами, включенными в модель.

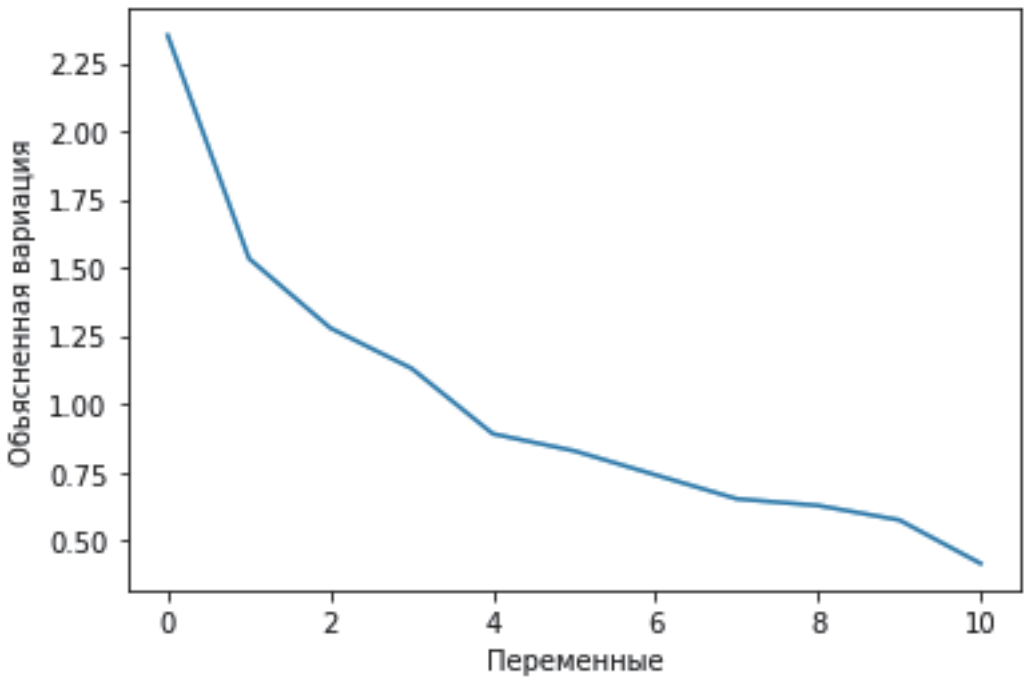


Рисунок 2.33 Визуализация результатов модели PCA (график каменистой осыпи PCA)

Первая переменная объясняет, приблизительно 25% информации, вторая переменная добавляет порядка 16%, третья около 13%, четвертая около 12%, пятая около 9%, шестая около 8%, седьмая около 7,5%, восьмая около 7%, 9-11 переменные добавляют уже минимальный процент объяснения. Перегиб на графике показывает, что первые 5 переменных могут содержать большую часть информации, представленной в данных.

Далее выведем компонент PCA во фрейме данных Pandas.

```
pd.DataFrame(results.components_, columns=list(
    ['lot size',
    '#bedroom',
    '#bath',
    '#stories',
    'driveway',
    'rec room',
    'basement',
    'gas',
    'air cond',
    '#garage',
    'desire loc']))
```

Прогнозирование стоимости объекта недвижимости до применения анализа главных компонент. Для оценки вероятности применим гауссово распределение, а для прогнозирования наивный классификатор Байеса.

```
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
import pylab as plt
# Выбор наивного классификатора Байеса; для оценки вероятности применяется гауссово распределение
gnb = GaussianNB()
# Подгонка данных
fit = gnb.fit(x1, y1)
# Прогнозирование для неизвестных данных
pred = fit.predict(x1)
# Изучение матрицы несоответствий
print(confusion_matrix(pred,y1))
# Подсчет правильно классифицированных случаев
print(confusion_matrix(pred,y1).trace())
```

Рисунок 2.34 Построение прогноза и матрицы несоответствий

Получим матрицу несоответствий и правильность классифицированных случаев.

```
[[0 0 0 ... 0 0 0]
 [0 1 0 ... 0 0 0]
 [0 0 1 ... 0 0 0]
 ...
 [0 0 0 ... 1 0 0]
 [0 0 0 ... 0 1 0]
 [0 0 0 ... 0 0 1]]
196
```

Рисунок 2.35 Матрица несоответствий

Как видно из рисунка, после проверки матрицы несоответствий были просуммированы все элементы диагонали. Так, наивный классификатор Байеса выдает 196 правильных прогнозов из 546, то есть прогностическая сила составляет 35,9%.

Далее проведем прогнозирование стоимости недвижимости с наращиванием количества главных компонент.

```
# Массив будет заполнен правильно спрогнозированными наблюдениями
predicted_correct = []
for i in range(1,10): # Перебор первых 10 обнаруженных главных компонент
    model = PCA(n_components = i) # Создание экземпляра модели PCA с разным количеством компонентов
    results = model.fit(x1) # Подгонка модели PCA по x=переменным
    Z = results.transform(x1) # Z содержит результат в форме матрицы
    fit = gnb.fit(Z,y1) # Применение наивного классификатора Байеса с гауссовым распределением для оценки
    pred = fit.predict(Z) # Собственно прогнозирование с использованием подогнанной модели
    # В конце каждой итерации добавляется новое количество правильно классифицированных наблюдений
    predicted_correct.append(confusion_matrix(pred,y1).trace())
    print(predicted_correct) # Выводя этот массив, видно добавление новых наблюдений
plt.plot(predicted_correct) # Результат в графической форме
plt.show() # Вывод графика
```

Рисунок 2.36 Реализация прогнозирования с наращиванием главных компонент

Далее визуализируем зависимость прогностической силы модели от количества добавляемых в модель переменных.

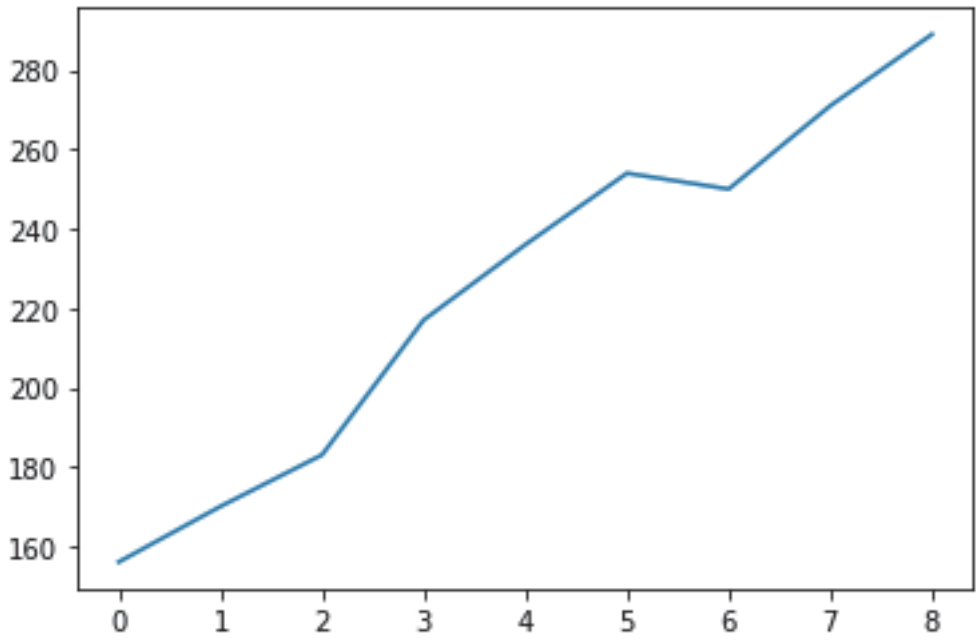


Рисунок 2.37 Прогностическая сила модели при 9 скрытых переменных

По графику видно, что добавление новых скрытых переменных в модель сильно повышает прогностическую способность.

Всего с 4 скрытыми переменными классификатор лучше справляется с прогнозированием стоимости объекта недвижимости, чем с 11 исходными. Также добавление скрытых переменных свыше 5 увеличивает прогностическую способность. Это показывает, что количество скрытых переменных для увеличения прогностической силы может быть увеличено.

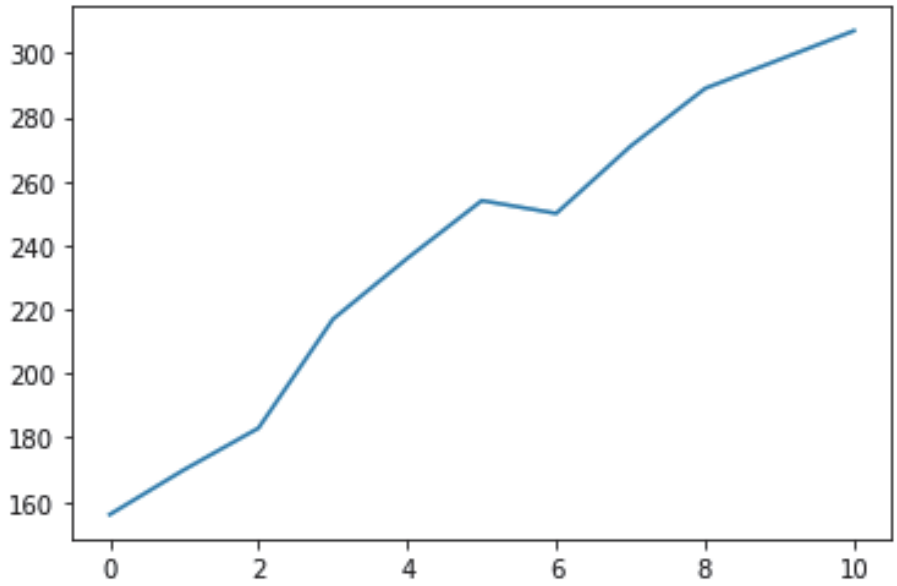


Рисунок 2.38 Прогностическая сила модели при 11 скрытых переменных

При 11 скрытых переменных прогностическая сила модели превышает значение 300, то есть 55%. Что говорит о том, что наращивание и поиск скрытых переменных будет прибавлять прогностической силе модели, хотя и не в такой силе как при первых 5-6 переменных.

Самостоятельная работа

Используя данные файла Rural_prices.csv или любой иной подходящий файл из предыдущих задач, постройте метод PCA и выведите процент объясненной вариации для 11 компонент и визуализируйте результат для первых 2 компонент.

1. Необходимо использовать следующий набор библиотек:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
import matplotlib.pyplot as plt
```

2. Стандартизировать данные.

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(x1)
```

3. Применить PCA.

```
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)
```

4. Вывести и просмотреть доли объясненной дисперсии для 11 компонент.

```
explained_variance = pca.explained_variance_ratio_
print(f'Объясненная дисперсия первой главной компоненты:
{explained_variance[0]:.2%}')
print(f'Объясненная дисперсия второй главной компоненты:
{explained_variance[1]:.2%}')
```

5. Визуализировать результаты PCA для 2 первых компонент.

```
plt.figure(figsize=(8, 6))
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y1, cmap='viridis')
plt.colorbar().set_label('')
plt.xlabel('Первая главная компонента')
plt.ylabel('Вторая главная компонента')
plt.title('Результаты PCA для набора данных Rural_prices')
plt.show()
```

Вопросы для самоконтроля

1. Какое основное назначение метода главных компонент?
2. Как вычисляется прогностическая сила модели?

3. Какова роль наивного Байевского классификатора в реализации метода PCA?

4. Для каких целей в работе применялось гауссово распределение?

5. Что является признаком усиления прогностической силы модели при добавлении новых скрытых переменных?

Лабораторная работа № 2.5
«Реализация классификатора текстов на основе модели KNN»

Теоретические положения

Машинное обучение – технологии автоматического обучения алгоритмов искусственного интеллекта по распознаванию и классификации на тестовых выборках объектов.

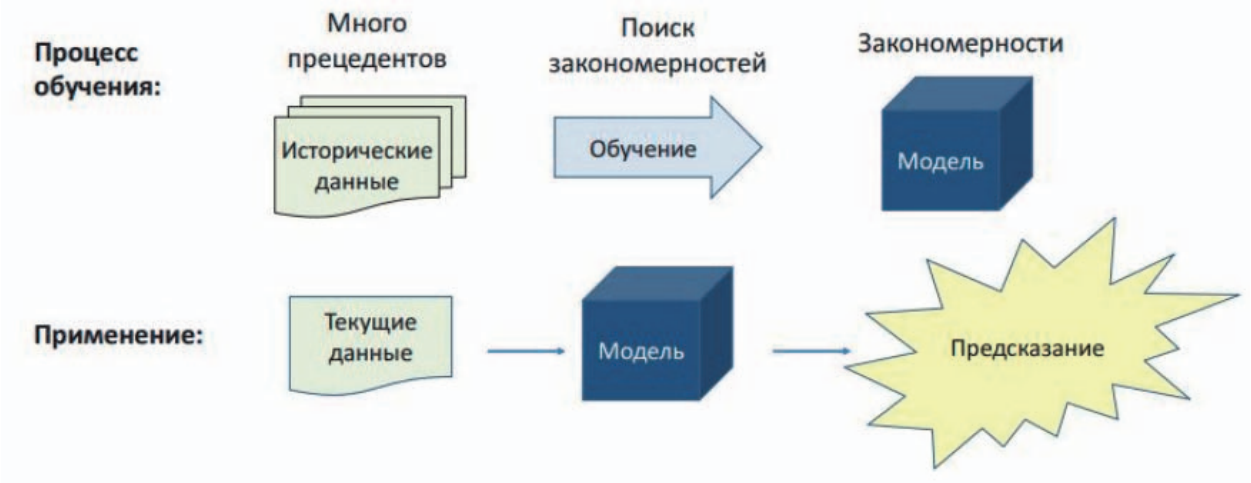


Рисунок 2.39 Ключевые этапы машинного обучения

К основным задачам машинного обучения относятся: регрессия, классификация и кластеризация [10, 12]. В анализе данных **классификация** – разбиение множества объектов или наблюдений на априорно заданные группы, называемые классами, внутри каждой из которых они предполагаются похожими друг на друга, имеющими примерно одинаковые свойства и признаки. В статистике и анализе данных **классом** называют группу объектов или явлений, обладающих общими свойствами. Например, среди заемщиков банка можно выделить классы добросовестных (которые не допускают просрочки) и недобросовестных (допускают). Также клиентов можно разбить на классы по уровню их активности (активный, пассивный) и т.д.

Виды классов:

- непересекающиеся – одно наблюдение может одновременно принадлежать только одному классу
- пересекающиеся – одно и то же наблюдение может принадлежать нескольким классам одновременно

- нечеткими – наблюдение принадлежит к классу с некоторой степенью принадлежности (обычно степень принадлежности задается в интервале от 0 до 1).

Метод k-ближайших соседей используется для решения задачи классификации. Он относит объекты к классу, которому принадлежит большинство из k его ближайших соседей в многомерном пространстве признаков. Число k – это количество соседних объектов в пространстве признаков, которые сравниваются с классифицируемым объектом. Иными словами, если k = 10, то каждый объект сравнивается с 10-ю соседями.

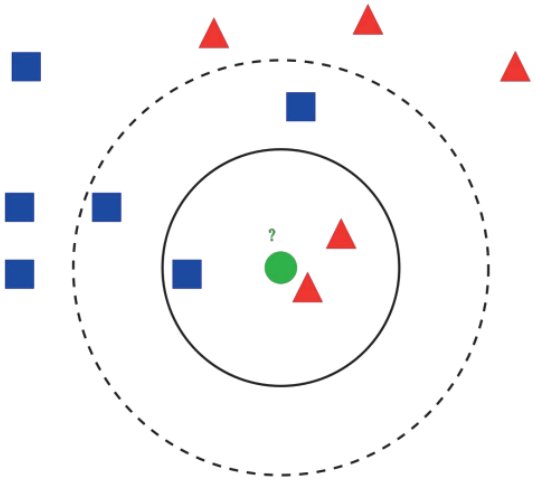


Рисунок 2.40 Метод k-ближайших соседей

На первом шаге алгоритма следует задать число k – количество ближайших соседей. Если принять k = 1, то алгоритм потеряет обобщающую способность (то есть способность выдавать правильный результат для данных, не встречавшихся ранее в алгоритме) так как новой записи будет присвоен класс самой близкой к ней. На втором шаге находятся k записей с минимальным расстоянием до вектора признаков нового объекта (поиск соседей). Для упорядоченных значений атрибутов находится Евклидово расстояние:

$$D_E = \sqrt{\sum_i^n (x_i - y_i)^2}, \quad (2.5)$$

где n – количество атрибутов.

При нахождении расстояния иногда учитывают значимость атрибутов. Она определяется экспертом или аналитиком субъективно, полагаясь на собственный опыт. В таком случае при нахождении расстояния каждый i-ый квадрат разности в сумме умножается на коэффициент Zi. Например, если атрибут A в три раза важнее атрибута B (ZA = 3, ZB = 1), то расстояние будет находиться следующим образом:

$$D_E = \sqrt{3(x_A - y_A)^2 + (x_B - y_B)^2}, \quad (2.6)$$

Подобный прием называют растяжением осей (stretching the axes), что позволяет снизить ошибку классификации. На следующем шаге, когда найдены записи, наиболее похожие на новую, необходимо решить, как они

влияют на класс новой записи. Для этого используется функция сочетания (combination function). Одним из основных вариантов такой функции является простое невзвешенное голосование (simple unweighted voting).

Простое невзвешенное голосование. Расстояние от каждой записи при голосовании здесь больше не играет роли. Все имеют равные права в определении класса. Каждая имеющаяся запись голосует за класс, к которому принадлежит. Новой записи присваивается класс, набравший наибольшее количество голосов. Но что делать в случае, если несколько классов набрали равное количество голосов? Эту проблему снимает взвешенное голосование (weighted voting).

Взвешенное голосование. Учитывается расстояние до новой записи. Чем меньше расстояние, тем более значимый вклад вносит голос. Голоса за класс находятся по следующей формуле:

$$votes\ (class) = \sum_{i=1}^{n} \frac{1}{d^2(X,Y_i)}, \quad (2.7)$$

где d2(X, Yi) – квадрат расстояния от известной записи Yi до новой X, n – количество известных записей класса, для которого рассчитываются голоса, class - наименование класса.

Нормализация. Разные атрибуты могут иметь разный диапазон представленных значений в выборке (например, атрибут А представлен в диапазоне от 0.1 до 0,5, а атрибут Б представлен в диапазоне от 1000 до 5000), то значения дистанции могут сильно зависеть от атрибутов с большими диапазонами. Нормализация предполагает замену номинальных признаков так, чтобы каждый из них лежал в диапазоне от 0 до 1.

Минимаксная нормализация:

$$X^x = \frac{X - \min(X)}{\max(X) - \min(X)}, \quad (2.8)$$

Отбор признаков. Важным при решении задачи является умение правильно отобрать и даже создать признаки. В англоязычной литературе это называется Feature Selection и Feature Engineering. В то время как Future Engineering довольно творческий процесс и полагается больше на интуицию и экспертные знания, для Feature Selection есть уже большое количество готовых алгоритмов. Популярным примером классификации в интернете является задача про классификацию ирисов.

Библиотека Scikit-Learn – это Python-библиотека, впервые разработанная David Cournapeau в 2007 году [2,3]. В этой библиотеке находится большое количество алгоритмов для задач, связанных с классификацией и машинным обучением в целом. Scikit-Learn даёт доступ ко множеству различных алгоритмов классификации. Вот основные из них: Метод k-ближайших соседей (K-Nearest Neighbors); Метод опорных векторов (Support Vector Machines); Классификатор дерева решений (Decision Tree Classifier) / Случайный лес (Random Forests); Наивный байесовский метод (Naive Bayes); Линейный дискриминантный анализ (Linear Discriminant Analysis); Логистическая регрессия (Logistic Regression).

knn.py

```python
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

iris_dataset = load_iris()

X_train, X_test, y_train, y_test = train_test_split(iris_dataset["data"], iris_dataset["target"])

model = KNeighborsClassifier(n_neighbors=4)     # количество соседей
model.fit(X_train, y_train)

expected = y_test
predicted = model.predict(X_test)

print(metrics.classification_report(y_test, predicted))
print(metrics.confusion_matrix(y_test, predicted))
```

Рисунок 2.41 Пример реализации алгоритма knn в Python

Матрица ошибок. Допустим, что у нас есть два класса и алгоритм, предсказывающий принадлежность каждого объекта одному из классов, тогда матрица ошибок классификации будет выглядеть следующим образом:

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Рисунок 2.42 Матрица ошибок

На рисунке $\hat{y}$ – это ответ алгоритма на объекте, а y – истинная метка класса на этом объекте. Таким образом, ошибки классификации бывают двух видов: False Negative (FN) и False Positive (FP).

13	0	0
0	11	1
0	2	11

Рисунок 2.43 Матрица ошибок классификации ириса

Количество верно классифицированных ирисов равно 35 единиц. Неверно классифицировано 3 ириса.

Доля правильных ответов модели (Accuracy). Интуитивно понятной, очевидной и почти неиспользуемой метрикой является accuracy – доля правильных ответов алгоритма:

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad (2.9)$$

Ограничение Accuracy – Метрика бесполезна в задачах с неравными классами.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	13
1	0.85	0.92	0.88	12
2	0.92	0.85	0.88	13
accuracy			0.92	38
macro avg	0.92	0.92	0.92	38
weighted avg	0.92	0.92	0.92	38

Рисунок 2.44 Пример отчета классификации

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными.

$$precision = \frac{TP}{TP+FP} \quad (2.10)$$

Recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.

$$recall = \frac{TP}{TP+FN} \quad (2.11)$$

Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок.

F-мера - среднее гармоническое precision и recall.

$$F_b = (1 + b^2) * \frac{precision*recall}{(b^2*precision)+ recall} \quad (2.12)$$

Преимущества алгоритма. Алгоритм устойчив к аномальным выбросам, так как вероятность попадания такой записи в число k-ближайших соседей мала. Если же это произошло, то влияние на голосование (особенно взвешенное) (при k>2) также, скорее всего, будет незначительным, и, следовательно, малым будет и влияние на итог классификации. Программная реализация алгоритма относительно проста. Результат работы алгоритма легко поддается интерпретации. Экспертам в различных областях вполне понятна логика работы алгоритма, основанная на нахождении схожих объектов. Возможность модификации алгоритма, путем использования

наиболее подходящих функций сочетания и метрик позволяет подстроить алгоритм под конкретную задачу.

Классификация текстов. Показатель TF-IDF оценивает значимость слова в документе, на основе данных о всей коллекции документов. Аббревиатура расшифровывается так: TF – term frequency (частота слова), а IDF – inverse document frequency (обратная частота документа). Это простой и удобный способ оценить важность термина для какого-либо документа относительно всех остальных документов. Принцип такой – если слово встречается в каком-либо документе часто, при этом встречаясь редко во всех остальных документах – это слово имеет большую значимость для того самого документа.

Векторная модель (англ. vector space model) – в информационном поиске представление коллекции документов векторами из одного общего для всей коллекции векторного пространства. Мера TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов (количество слов набора определяет размерность вектора) в каждом документе. Подобная модель называется векторной моделью и даёт возможность сравнивать тексты, сравнивая представляющие их вектора [8].

TF – это частотность термина, которая измеряет, насколько часто термин встречается в документе. Логично предположить, что в длинных документах термин может встретиться в больших количествах, чем в коротких, поэтому абсолютные числа тут не подходят. Поэтому применяют относительные – делят количество раз, когда нужный термин встретился в тексте, на общее количество слов в тексте.

$$tf(t,d) = \frac{n_t}{\sum kn_k} \quad 2.13)$$

Показатель IDF равен логарифму отношения количества документов в коллекции к количеству документов в коллекции, в которых встречается заданное слово.

$$idf(t,D) = log\frac{|D|}{|\{d_i \in D : t \in d_i\}|} \quad (2.14)$$

Учет IDF позволяет снизить вес слов, употребляемых часто (например, предлогов).

Большой вес в TF-IDF получат слова с высокой частотой в пределах конкретного документа и с низкой частотой употреблений в других документах.

$$\text{Tf-idf(t,d,D)} = \text{tf(t,d)} * \text{idf(t,D)} \quad (2.15)$$

Пример расчета TF-IDF 1. Возьмём документ из 10000 символов в котором слово “пропан” встречается 25 раз, а коллекция состоит из 2 миллионов документов, в 2000 из которых также встречается данное слово.

TF = 25 ÷ 10000 = 0,0025
IDF = lg(2000 ÷ 2000000) = lg(1 ÷ 1000) = -3 (lg - логарифм с основанием 10)

TF-IDF = 0,0025 × 3 = 0,0075.

Пример расчета TF-IDF 2. Возьмём тот же документ из 10000 символов в котором союз “но” встречается 30 раз, а коллекция по-прежнему состоит из 2 миллионов документов, в 200000 из которых также встречается данное слово.

TF = 30 ÷ 10000 = 0,003
IDF = lg(200000 ÷ 2000000) = lg(1 ÷ 10) = -1 (lg - логарифм с основанием 10)
TF-IDF = 0,003 × 1 = 0,003.

Задание:

Условие. Имеется датасет, состоящий из 45 статей, по 15 статей на каждую рубрику (файл «Data.zip). Каждую статью необходимо представить в виде вектора чисел (TF-IDF).

Требуется реализовать классификатор текстов на основе модели KNN (К-ближайших соседей). Для этого необходимо обучить KNN модель на полученных векторах чисел. При загрузке в модель статьи (не из обучающей выборки) – на выходе получаем рубрику (класс), к которой она относится.

Перед тем, как рассчитывать TF-IDF нужно из статей:
- удалить стоп слова (https://snipp.ru/seo/stop-ru-words)
- удалить именованные сущности и цифры (библиотека natasha)
- от каждого слова взять только стемму/лемму (библиотека pymorphy).

При начале работы над заданием рекомендуется ознакомиться с документацией библиотеки scikit-learn [9,10,11].

Данные: https://disk.yandex.ru/d/Bqdbf9gIGJDNHw

Пример реализации: https://disk.yandex.ru/d/BvFErvhz9JtCsg

Методические указания к выполнению лабораторной работы

Необходимо установить библиотеку natasha (pip install natasha).

1. Далее подключаем библиотеки.

```
import os               # Управление операционной системой
import pandas as pd     # Data frames
import re               # Разбиение текста на слова
import numpy as np      # Список в виде массива
```

2. Импортируем инструмент векторизации.

```
from sklearn.feature_extraction.text import TfidfVectorizer
```

3. Определим пути ко всем файлам с исходными данными.

```
# Пути основные
cwd = os.getcwd()             # Текущая директория
```

```
cwd_data = cwd + '\Data\\'  # Путь до директории с исходными данными

# Пути дополнительные
filePath_stopWords = cwd + '\\' + 'stop-words.txt'  # Путь к файлу со стоп-словами, скаченному по ссылке: https://snipp.ru/seo/stop-ru-words

# Пути к папкам с рубриками
foldersNames_list = os.listdir(cwd_data)  # Список папок с рубриками
rubricsCount = len(foldersNames_list)     # Число рубрик
foldersPaths_list = []                    # Список путей к папкам с рубриками
for i in range(rubricsCount):
    foldersPaths_list.append(cwd_data + foldersNames_list[i])

# Функция получения путей ко всем статьям в папке рубрики
def getFilesPathsInRubricFolder(folderPath_rubric):
    """
    folderPath_rubric - Путь к папке рубрики
    """
    # Файлы в папке рубрики
    filesNames_rubric_list = os.listdir(folderPath_rubric)  # Названия файлов со статьями в папке рубрики
    filesCount = len(filesNames_rubric_list)                # Число файлов (статей) в папке рубрики

    # Пути ко всем статьям в папке рубрики
    filesPaths_rubric_list = [] # Список путей ко всем статьям в папке рубрики
    for i in range(filesCount):
        filesPaths_rubric_list.append(folderPath_rubric + '\\' + filesNames_rubric_list[i])

    return filesPaths_rubric_list

# Пути ко всем статьям в папках рубрик
foldersPaths_all = []             # Пути ко всем статьям в папках рубрик
foldersPaths_all_rubrics = []     # Рубрика, соответствующая пути к статье (для создания таблицы Рубрика:Путь_к_статье)
for i in range(rubricsCount):

    foldersPaths_inRubric = getFilesPathsInRubricFolder(folderPath_rubric = foldersPaths_list[i])
```

```
    foldersPaths_all += foldersPaths_inRubric

    arcticlesCount_inRubric = len(foldersPaths_inRubric)

    for j in range(arcticlesCount_inRubric):
        foldersPaths_all_rubrics.append(foldersNames_list[i])

# Таблица [Рубрика, Путь_к_статье]
df_rubricsAndArticlesPaths_all = pd.DataFrame({'Rubric' :
foldersPaths_all_rubrics,
                                'ArticlePath' : foldersPaths_all})

articlesCount = len(foldersPaths_all) # Число статей всего

# Пути к подготовленным текстам
# Путь к папке с подготовленными текстами
folderPath_preparedData = cwd + '\\' + 'Data_prepared'

# Пути к папкам рубрик
folderPath_preparedData_folders = os.listdir(folderPath_preparedData) #
Список папок
foldersPaths_preparedData_rubrics = [] # Пути к папкам рубрикам с
подготовленными текстами
for i in range(rubricsCount):
    path = folderPath_preparedData + '\\' + foldersNames_list[i]
    foldersPaths_preparedData_rubrics.append(path)

# Пути ко всем статьям в папках рубрик
foldersPaths_preparedData_all = []          # Пути ко всем статьям в папках
рубрик (подготовленные тексты)
for i in range(articlesCount):
    path = foldersPaths_all[i].replace('TextClassifier\\Data',
'TextClassifier\\Data_prepared', 1)
    foldersPaths_preparedData_all.append(path)
```

4. Подготовим данные.

```
# Импорт исходных данных
# Функция считывания текста из файла txt (+ удаление пустых строк, +
простая очистка)
def readTextFromFile(filePath):
```

```
    # Считывание текста из файла построчно
    with open(filePath, 'r', encoding = 'utf8') as f:
        contents_list = f.readlines()      # Строки в файле

    # Объединение строк в единый текст и очистка
    text = ''                              # Текст из файла
    for i in range(len(contents_list)):
        text += contents_list[i]           # Объединение

        # Очистка от знаков препинания
        text = text.replace('.', '')
        text = text.replace(',', '')
        text = text.replace(' — ', ' ')
        text = text.replace(' - ', ' ')

        text = text.replace('\n', ' ')     # Очистка от символа \n
        text = text.replace('  ', ' ')     # Очистка от двойных пробелов


    return text

# Список текстов
articlesTexts_list = []
for i in range(articlesCount):
    filePath = df_rubricsAndArticlesPaths_all.iloc[i, 1]
    text = readTextFromFile(filePath = filePath)
    articlesTexts_list.append(text)

# Подготовка текстов
# 1. Удаление стоп-слов
stopWords_list = [] # Список стоп-слов

# Считывание текста из файла со стоп-словами построчно
with open(filePath_stopWords, 'r', encoding = 'utf8') as f:
    stopWords_list = f.readlines()     # Строки в файле
# Очистка от символа \n
stopWords_list_len = len(stopWords_list) # Число стоп-слов
for i in range(stopWords_list_len):
    stopWords_list[i] = stopWords_list[i].replace('\n', '')

# Удаление стоп-слов
```

```
text_words_list = re.findall(r'\b\S+\b', articlesTexts_list[0]) # Список всех слов из текста (r'\b\S+\b' - регулярное (шаблонное) выражение, где \b - граница слова, \S - непробельный знак, \S+ - любая последовательность \S )
text_words_list_withoutStopWords = [] # Список всех слов из текста, кроме стоп-слов
wordsCount_inText = len(text_words_list)
for i in range(wordsCount_inText):

    word = text_words_list[i].lower() # Слово (.lower() - с маленькой буквы)

    # Если слово НЕ входит в список стоп-слов, то сохранение в список text_words_list_withoutStopWords
    if not word in stopWords_list:
        text_words_list_withoutStopWords.append(text_words_list[i])
# 2. удалить именованные сущности и цифры (библиотека natasha)
# https://habr.com/ru/post/516098/

# Подключение элементов библиотеки natasha
from natasha import (
    Segmenter,

    NewsEmbedding,
    NewsMorphTagger,
    NewsSyntaxParser,

    Doc
)
from natasha import NewsNERTagger
from natasha import MorphVocab
morph_vocab = MorphVocab()
# Функция полученя подготовленных текстов (выполнение 3-х видов очистки)
def getPreparedTexts(textInitial):
    """
    Parameters:
    textInitial - Исходный текст

    Returns:
    text_prepared_1 - Подготовленный текст после 1 вида очитски;
    text_prepared_2 - Подготовленный текст после 2 вида очитски;
```

```
    text_prepared_3 - Подготовленный текст после 3 вида очитски.

    """

    textInitial_words_list = re.findall(r'\b\S+\b', textInitial) # Список всех слов в тексте исходном (r'\b\S+\b' - регулярное (шаблонное) выражение, где \b - граница слова, \S - непробельный знак, \S+ - любая последовательность \S )
    textInitial_wordsCount = len(textInitial_words_list)        # Число слов в тексте исходном

    # 1. Удаление стоп-слов
    text_prepared_1 = []                    # Подготовленный текст после 1 вида очистки
    textInitial_words_list_withoutStopWords = [] # Список всех слов из текста, кроме стоп-слов
    for i in range(textInitial_wordsCount):
        word = textInitial_words_list[i].lower()        # Слово (.lower() - с маленькой буквы)
        # Если слово НЕ входит в список стоп-слов, то сохранение в список text_words_list_withoutStopWords
        if not word in stopWords_list:

textInitial_words_list_withoutStopWords.append(textInitial_words_list[i])
    text_prepared_1_list = textInitial_words_list_withoutStopWords
    # Представлние списка слов в виде единого текста
    text_prepared_1 = ' '.join(word for word in text_prepared_1_list)

        # 2. Удаление именованных сущностей и цифр (библиотека natasha)
    # https://habr.com/ru/post/516098/
    text_prepared_2 = []                    # Подготовленный текст после 1 вида очистки

    # 2.1 Удаление именованных сущностей
    text_prepared_2_1 = []

    segmenter = Segmenter()

    emb = NewsEmbedding()

    morph_tagger = NewsMorphTagger(emb)
```

```
    syntax_parser = NewsSyntaxParser(emb)

        #text = '"Сотрудники филиала "Россети Московский регион" – "Новая Москва" выдали 1059,6 киловатта мощности общеобразовательной школе на 1,1 тысячи мест и детскому саду на 350 мест. Для подключения к сети и выдачи необходимой мощности специалисты построили блочную комплектную трансформаторную подстанцию, а также проложили 12 кабельных линий 0,4 киловольта общей протяженностью два километра", - отмечается в нем.'
    text = text_prepared_1
    doc = Doc(text)

    doc.segment(segmenter)
    doc.tag_morph(morph_tagger)
    doc.parse_syntax(syntax_parser)


    #from natasha import NewsNERTagger
    ner_tagger = NewsNERTagger(emb)
    doc.tag_ner(ner_tagger)
    #doc.ner.print()

    #from natasha import MorphVocab
    morph_vocab = MorphVocab()

    # Поиск именованных сущностей
    namedEntities_list = []              # Именованные сущности в тексте
    namedEntities_count = len(doc.spans)  # Число именованных сущностей в тексте
    for i in range(namedEntities_count):
        namedEntities_list.append(doc.spans[i].text)

    # Удаление именованных сущностей
    text1 = text
    for i in range(namedEntities_count):
        text1 = text1.replace(namedEntities_list[i], '')

    # Удаление лишних дйоных и тройных пробелов
    text1 = text1.replace('  ', ' ')
    text1 = text1.replace('  ', ' ')
```

```
    text_prepared_2_1 = text1


    # 2.2 Поиск и удаление цифр
    text_prepared_2_1_words_list = re.findall(r'\b\S+\b', text_prepared_2_1) #
Список всех слов в тексте исходном (r'\b\S+\b' - регулярное (шаблонное)
выражение, где \b - граница слова, \S - непробельный знак, \S+ - любая
последовательность \S )
    text_prepared_2_1_wordsCount = len(text_prepared_2_1_words_list)
# Число слов в тексте исходном
    text_words_list_withoutNumbers = [] # Список всех слов из текста,
кроме цифр
    for i in range(text_prepared_2_1_wordsCount):

        word = text_prepared_2_1_words_list[i] # Слово

        # Если слово НЕ равно числу, то сохранение в список
text_words_list_withoutNumbers
        if word.isdigit() != True:
            text_words_list_withoutNumbers.append(word)

    text_prepared_2_list = text_words_list_withoutNumbers
    # Представлние списка слов в виде единого текста
    text_prepared_2 = ' '.join(word for word in text_prepared_2_list)

    # 3. Лемматизация (лемма - начальная форма слова)
    # от каждого слова взять только стемму/лемму (библиотека pymorphy)
    #https://www.youtube.com/watch?v=Tyk0aHHzlYE
    text_prepared_3 = []                    # Подготовленный текст после 3
вида очистки

    # Подготовка текста к библиотеке natasha
    doc = Doc(text_prepared_2)
    doc.segment(segmenter)
    doc.tag_morph(morph_tagger)

    # Лемматизация (определение леммы для каждого слова)
    for token in doc.tokens:
        token.lemmatize(morph_vocab)
    dict_wordAndLemma = {_.text: _.lemma for _ in doc.tokens}   # Словарь
[Слово:Лемма]
```

```
    text_prepared_3_list = list(dict_wordAndLemma.values())     # Список
лемм
    # Представлние списка слов (лемм) в виде единого текста
    text_prepared_3 = ' '.join(word for word in text_prepared_3_list)



    return text_prepared_1, text_prepared_2, text_prepared_3 # ,
text_prepared_2_1

# Получение подготовленных текстов (выполнение 3-х видов очистки)
texts_prepared_1 = [] # Подготовленные тексты после 1 вида очистки
texts_prepared_2 = [] # Подготовленные тексты после 2 вида очистки
texts_prepared_3 = [] # Подготовленные тексты после 3 вида очистки
for i in range(articlesCount):
    prepText_1, prepText_2, prepText_3 = getPreparedTexts(textInitial =
articlesTexts_list[i])
    texts_prepared_1.append(prepText_1)
    texts_prepared_2.append(prepText_2)
    texts_prepared_3.append(prepText_3)
    print(i)
# Таблица [Рубрика, Текст_статьи, Текст_очищенный_1,
Текст_очищенный_2, Текст_очищенный_3]
df_main = pd.DataFrame({'Rubric' : foldersPaths_all_rubrics,
                        'Text' : articlesTexts_list,
                        'Text_prep_1' : texts_prepared_1,
                        'Text_prep_2' : texts_prepared_2,
                        'Text_prep_3' : texts_prepared_3})

#
# Сохранение подготовленных текстов
#
# Создание папки, если не существует
folderPath_preparedData = cwd + '\\' + 'Data_prepared' # Путь к папке с
подготовленными текстами
if not 'Data_prepared' in os.listdir(cwd):
    os.mkdir(folderPath_preparedData) # Создание папки

# Создание папок рубрик, если не существуют
for i in range(rubricsCount):
    if not foldersNames_list[i] in folderPath_preparedData_folders:
        os.mkdir(foldersPaths_preparedData_rubrics[i]) # Создание папки
```

```
# Сохранение подготовленных текстов
for i in range(articlesCount):
    with open(foldersPaths_preparedData_all[i], 'w', encoding = 'utf8') as
text_file:
        text_file.write(texts_prepared_3[i])
```

5. Модель KNN

```
#
# Импорт подготовленных текстов
#
# Список текстов подготовленных
articlesTexts_prepared_list = []
for i in range(articlesCount):
    with open(foldersPaths_preparedData_all[i], 'r', encoding = 'utf8') as
text_file:
        contents_list = text_file.readlines()              # Текст в файле
(построчно)
        articlesTexts_prepared_list.append(contents_list[0])   # Сохранение
текста в список

# Список имен файлов с текстами
articlesTexts_prepared_filesNames = []
for i in range(articlesCount):
    fileName = foldersPaths_preparedData_all[i]
    fileName = fileName[fileName.rfind('\\') + 1 :]
    articlesTexts_prepared_filesNames.append(fileName)

# Номера рубрик
rubricsNumbers_list = []        # Норера рубрик для всех статей (n = числу
статей)
for i in range(articlesCount):
    rubricName = df_rubricsAndArticlesPaths_all.iloc[i, 0]
    # Поиск номера рубрики
    for j in range(rubricsCount):
        if rubricName == foldersNames_list[j]:
            rubricNumber = j
    rubricsNumbers_list.append(rubricNumber)

# Таблица [Номер_рубрикик, Рубрика, Файл_текста,
Текст_подготовленный]
df_preparedTexts = pd.DataFrame({'RubricNumber' : rubricsNumbers_list,
```

```
                    'Rubric' : foldersPaths_all_rubrics,
                    'FileName' : articlesTexts_prepared_filesNames,
                    'TextPrep' : articlesTexts_prepared_list})

#
# Разделение всех статей на выборки Обучающую и Тестовую
#
# Представление текстов в виде числовых векторов
numericVectors_mx = 0 # Тексты в виде числоых векторов. Матрица
размером: [Число_документов x Число_уникальных_слов_всего]
docs = articlesTexts_prepared_list #    Документы (тексты)
vectorizer = TfidfVectorizer(sublinear_tf = True, max_df = 0.5, stop_words =
None)
res = vectorizer.fit_transform(docs)
res_words = vectorizer.get_feature_names()  # Список слов из всех текстов
res_mx = res.todense()                  # Тексты в виде числоых векторов.
Матрица размером: [Число_документов x
Число_уникальных_слов_всего]
res_idf = vectorizer.idf_

numericVectors_mx = res_mx

# Сохранение числовых векторов в таблицу df_preparedTexts
df_preparedTexts['NumericVector'] = list(numericVectors_mx)

# Обучение модели KNN
from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.datasets import load_iris

texts_dataset_data = numericVectors_mx              # Тексты в виде
числовых векторов (X)
texts_dataset_target = np.array(rubricsNumbers_list)  # Метки текстов
(номер рубрики для текста) (Y)
'''
# Для провекри на наборе данных iris_dataset
iris_dataset = load_iris()
iris_dataset_data = iris_dataset['data']
iris_dataset_target = iris_dataset['target']
'''
X_train, X_test, Y_train, Y_test = train_test_split(texts_dataset_data,
texts_dataset_target)
```

```
model_KNN = KNeighborsClassifier(n_neighbors = 4) # Число соседей
model_KNN.fit(X_train, Y_train)

predicted = model_KNN.predict(X_test)

print(metrics.classification_report(Y_test, predicted))
print(metrics.confusion_matrix(Y_test, predicted))

# Определение текста, который соответствует числовому вектору из тестовой выборки
idx_text = 0              # Индекс текста в таблице
for i in range(articlesCount):
    # Если вектор из таблицы df_preparedTexts равен вектору из тестовой выборки X_test, то сохранение индекса
    if (df_preparedTexts['NumericVector'][i] == X_test[0]).all():
        idx_text = i      # Индекс нужного текста в таблице
text_text = df_preparedTexts['TextPrep'][idx_text]

# Применение модели для первого текста тестовой выборки
predicted_text1 = model_KNN.predict(X_test[0])
print('\nПервый текст из тестовой выборки следующий:')
print(text_text)

print('\nПервый текст из тестовой выборки относится к следующей рубрике')
print('Номер рубрики: ', predicted_text1[0])
print('Название рубрики: ', foldersNames_list[predicted_text1[0]])
```

По итогам запуска скрипта мы должны получить следующие результаты.

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       0.80      1.00      0.89         4
           2       1.00      0.67      0.80         3

    accuracy                           0.92        12
   macro avg       0.93      0.89      0.90        12
weighted avg       0.93      0.92      0.91        12

[[5 0 0]
 [0 4 0]
 [0 1 2]]
```

Рисунок 2.45 Метрики построенной модели классификации

Видно, что модель классифицирует предложенные тексты с высоким уровнем точности.

```
Первый подготовленный текст из тестовой выборки следующий:
авиакомпания приостановить участие альянс сообщить пресс-служба перевозчик ограничение
международный авиасообщение сократиться бизнес-взаимодействие альянс ситуация совместный решение
апрель приостановить действие соглашение участие авиакомпания сообщение компания отметить
затронуть небольшой число путешественник причина существовать полетный ограничение s 7 ноябрь
сообщать членство млн пассажир совершить полет единый авиабилет авиакомпании-партнер ежегодно
альянс перевозить формировать трансферный поток перевозчик

Первый подготовленный текст из тестовой выборки относится к следующей рубрике
Номер рубрики:  0
Название рубрики:  Рубрика бизнес
```

Рисунок 2.46 Первое тестирование программы

Модель точно определила, что представленный текст является текстом относимым к рубрике «бизнес».

```
Первый подготовленный текст из тестовой выборки следующий:
полузащитник разговор корреспондент высказаться полуфинал сильный чемпионат чемпионат
сильный английский финал чемпион выигрывать испанский команда побить рекорд мостовая
полуфинал сыграть первый матч состояться апреля ответный встреча май

Первый подготовленный текст из тестовой выборки относится к следующей рубрике
Номер рубрики:  2
Название рубрики:  Рубрика спорт
```

Рисунок 2.47 Второе тестирование программы

Анализируя текст видим, что рубрика «спорт» определена точно. И, наконец, протестируем третью рубрику.

```
Первый подготовленный текст из тестовой выборки следующий:
творческий пространство проводить open call молодой художник дизайнер одежда тема
интерпретация образ женщина период история мода участвовать специалист возраст
победитель получить творческий мастерская контракт смочь принять участие групповой
выставка пространство тема фестиваль архстояние организатор мероприятие объявить
открытый конкурс художник архитектор дизайнер предлагать создать проект тема победитель
грант реализовать идея приоритет отдаваться пространственный инсталляциям коллективный
перформанс предполагать взаимодействие зритель

Первый подготовленный текст из тестовой выборки относится к следующей рубрике
Номер рубрики:  1
Название рубрики:  Рубрика культура
```

Рисунок 2.48 Третье тестирование программы

Рубрика «культура» также определена корректно.

Самостоятельная работа

1. Постройте классификатор для любых других трех категорий, например, «кулинария», «компьютерные игры», «high tech».
2. Реализуйте телеграмм-бот для построенной модели.

Вопросы для самоконтроля

1. Назовите основные задачи, решаемые с помощью машинного обучения.
2. Что такое классификация в решении задач машинного обучения?
3. Дай определение понятию «класс» в решении задач классификации.
4. Суть метода k-ближайших соседей.
5. Опиши алгоритм k-ближайших соседей.
6. Какие методы машинного обучения содержит библиотека sklearn?
7. Перечисли ключевые метрики качества моделей классификации.
8. Назначение и порядок расчета показателя TF-IDF.

Лабораторная работа № 2.6
«Классификация изображений на примере распознавания цифр»

Теоретические положения

Распознавание рукописных цифр – одно из классических примеров задач классификации изображений. Данная задача применяется в автоматическом чтении номеров банковских чеков, почтовых индексов и других документов.

Задача распознавания цифр состоит в том, чтобы по изображению цифры определить её значение. Например, если на вход подается изображение числа "7", система должна правильно классифицировать его как семёрку.

Для решения подобной задачи часто используется датасет MNIST, представляющий собой 60 тыс. обучающих и 10 тыс. текстовых изображений рукописных цифр одного размера. Каждое изображение представлено в виде матрицы чисел, где каждое число отражает интенсивность пикселя (от 0 до 255).

В решении задач классификации изображений применяются такие методы как: метод ближайших соседей, логистическая регрессия, а также широкое распространение получили нейронные сети.

Задание:

Условие. Имеется набор данных MNIST. Данные изображения расположены в пакете набора данных библиотеки Scikit-learn (изображения уже нормализированы, то есть масштабированы до единого размера 64x64).

Требуется: используя набор данных MNIST добиться того, чтобы компьютер распознавал числа.

Методические указания к выполнению лабораторной работы

1. Импортируем базу данных цифр из библиотеки Scikit-learn.

```
from sklearn.datasets import load_digits
import pylab as pl
```

2. Загрузим цифры. Указанная база данных содержит порядка 2 тыс. изображений, каждое из которых помечено наиболее вероятным значением цифры, представленной на картинке.

numbers = load_digits()

3. Использование библиотеки Scikit-learn. Изображение преобразуется в набор значений в оттенках серого.

pl.gray() # Изображение преобразуется в набор значений в оттенках серого

pl.matshow(numbers.images[0]) # Сначала выведем изображение

pl.show()

print(numbers.images[0]) # Вывод соответствующей матрицы

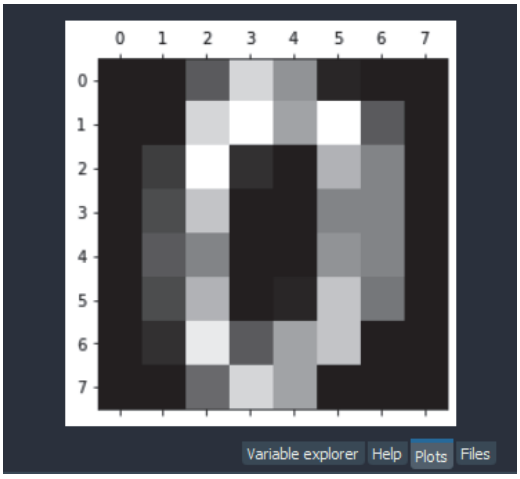


Рисунок 2.49 Матрица изображения цифры

```
[[ 0.  0.  5. 13.  9.  1.  0.  0.]
 [ 0.  0. 13. 15. 10. 15.  5.  0.]
 [ 0.  3. 15.  2.  0. 11.  8.  0.]
 [ 0.  4. 12.  0.  0.  8.  8.  0.]
 [ 0.  5.  8.  0.  0.  9.  8.  0.]
 [ 0.  4. 11.  0.  1. 12.  7.  0.]
 [ 0.  2. 14.  5. 10. 12.  0.  0.]
 [ 0.  0.  6. 13. 10.  0.  0.  0.]]

In [2]:
```

Рисунок 2.50 Матрица набора значений

Размытое изображение в оттенках серого преобразовалось в соответствующую матрицу набора значений. Если значение больше, то элемент ближе к белому, если значение меньше, то элемент ближе к черному. Так модель может определить интенсивность изображения и соответственно понять какая цифра изображена на картинке.

После работы с тренировочным набором и построения модели, далее передается тестовый набор и проверяется качество интерпретации модели.

4. Классификация изображений

#Импортируем требуемые библиотеки:

from sklearn.model_selection import train_test_split

```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import confusion_matrix
import pylab as plt

# Выберем целевую переменную
Y = numbers.target

# Подготовим данные
n_samples = len(numbers.images)

# С применением метода reshape преобразует матричную форму данных.
X = numbers.images.reshape((n_samples, -1))

# Разобьем данные на тестовый и тренировочный набор
X_train, X_test, y_train, y_test = train_test_split(X, Y, random_state=0)

# Выберем наивный классификатор Байеса, а для для оценки вероятности – распределение Гаусса
GB = GaussianNB()
# Подготовим данные
fit = GB.fit(X_train, y_train)
# Проведем прогнозирование по незнакомым данным
predicted_unknown = fit.predict(X_test)
# Создадим и выведем матрицу несоответствий
print(confusion_matrix(y_test, predicted_unknown))
```

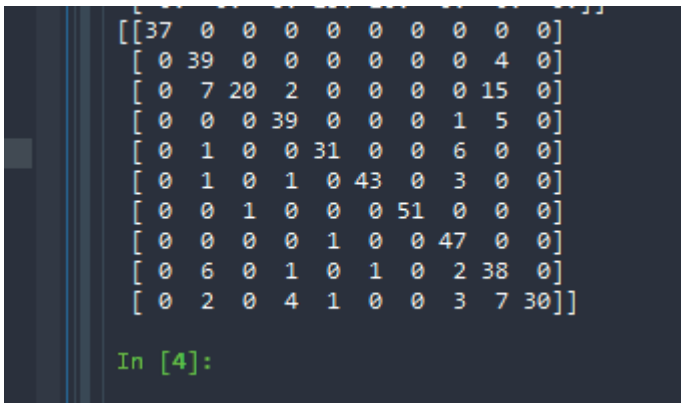


Рисунок 2.51 Матрица ошибок (несоответствий)

Матрица несоответствий – двумерный массив, показывающий насколько часто прогнозируемое число совпадало с верным. Увидеть это можно на главной диагонали. Так число 6 совпало с верным значением 52 раза.

Другие элементы матрицы показывают, сколько раз модель определяла значение по столбцу, хотя в реальности это было значение по строке (строки

и столбцы номеруются от 0 до 9 и указывают на соответствующую распознаваемую цифру. Так модель предсказывала 1 семь раз, хотя было изображено значение 2. (пересечение столбца 1 и строки 2). То есть, по большинству изображений прогнозы достаточно точны.

Существует критерий, по которому хорошей считается та модель, сумма значений по диагонали превышает суммы всех остальных значений матрицы.

5. Проведем сравнение прогнозов с реальными цифрами

```
# Сохраним матрицу изображения и прогнозов (в числовом виде) в одном массиве
images_and_predictions = list(zip(numbers.images, fit.predict(X)))
# Перебор первых 7 изображений
for index, (image, prediction) in enumerate(images_and_predictions[:7]):
    plt.subplot(7, 3, index + 6) # Добавление дополнительной поддиаграммы на сетке 7x3
    plt.axis('off') # Ось не отображается
    plt.imshow(image, cmap=plt.cm.gray_r, interpolation='nearest') # Изображение выводится в оттенках серого
    plt.title('Предсказание^ %i' % prediction) # Прогнозируемое значение выводится в заголовке изображения

# Выведем полную диаграмму, состоящую из 7 поддиаграмм.
plt.show()
```

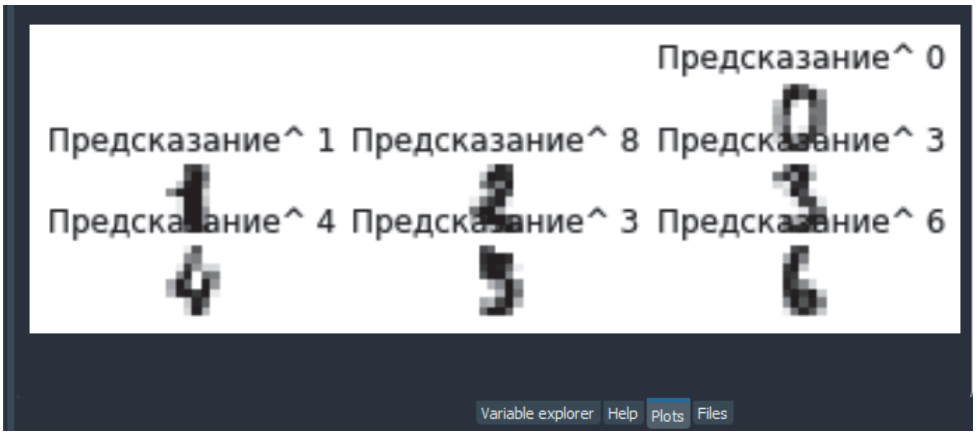


Рисунок 2.52 Результат классификации

Число 0 – верно распознано. 1 – верно. 2 – неверно (на изображении 2, а распознается как 8). 3 – верно. 4 – верно. 5 – неверно (хотя число изображено спорно). 6 – верно.

Самостоятельная работа

Используя предложенный в работе код, выведите результат классификации для всего ряда изображений цифр в диапазоне от 0 до 9 значений.

Вопросы для самоконтроля

1. Назовите библиотеку, которая содержит базу данных MNIST
2. Как должно быть преобразовано изображение для того, чтобы быть распознано алгоритмом классификации?
3. Что отражает матрица набора значений?
4. Какова точность модели классификации и по каким метрикам ее можно определить?
5. Какие методы применяются для решения задачи классификации изображений?
6. Какие задачи может решать модель классификации изображений?
7. Относится ли задача классификации цифр к задачам бинарной или многоклассовой классификации. Если да, то объясните почему?

Раздел 3
Автоматизация и отображение результатов анализа больших данных

Лабораторная работа № 3.1
«Построение веб-приложения модели машинного обучения в Streamlit»

Теоретические положения

Наиболее эффективным инструментом визуализации, отображения и автоматизации полученных результатов является построение веб-приложения анализа данных с применением библиотек Python. Наиболее подходящими библиотеками для решения подобных задач являются библиотеки Streamlit и Gradio.

Streamlit – библиотека Python для отображения результатов анализа данных в веб-приложении с возможностью упрощения дальнейшего пользования разработанными модулями анализа и автоматизации последнего при изменении или обновлении исходных данных [7].

Возможности данной библиотеки позволяют создавать многостраничное веб-приложение с функциями редактирования текста, отображения графиков, таблиц, виджетов и других объектов. В данной приложение могут быть загружены не только графики, но и изображения, видео, аудиозаписи.

Веб-приложение на streamlit может быть персонализировано с возможностью установления имя пользователя и пароля.

Streamlit имеет усиленные возможности по изменению шрифтов, цвета страницы, боковой панели и ее компонентов, применения встроенных компонентов визуализации и анализа данных, расширенные возможности компонентов API.

Посредством Streamlit Cloud созданное веб-приложение может быть выложено для совместной работы над проектом в облаке, что существенно повышает функциональность и производительность дальнейшего развития этого приложения. Таким образом, streamlit позволяет быстро и легко автоматизировать статистический анализ и применение методов машинного обучения. Изменяя, например, исходные данные вы можете достаточно быстро реализовать все компоненты необходимого анализа.

Задание:

Условие. Имеется построенная модель классификации ирисов и библиотека Streamlit.

Требуется: отобразить результаты построения модели классификации ирисов при помощи библиотеки Streamlit.

Методические указания к выполнению лабораторной работы

Перед началом построения приложения необходимо установить библиотеку через pip install Streamlit.

Далее создать файл с расширением .py и указать там первые параметры приложения: библиотеки, заголовок главной страницы и подпись боковой панели «Параметры для классификации».

```
import streamlit as st
import pandas as pd
from sklearn import datasets
from sklearn.ensemble import RandomForestClassifier

st.write("""
# Классификация цветов
Приложение классифицирует вид ириса по выбранным параметрам
""")
st.sidebar.header('Параметры для классификации')
```

Далее запустим приложение в Anaconda Prompt посредством команды Streamlit run путь к файлу file.py

В результате откроется главная страница будущего классификатора.

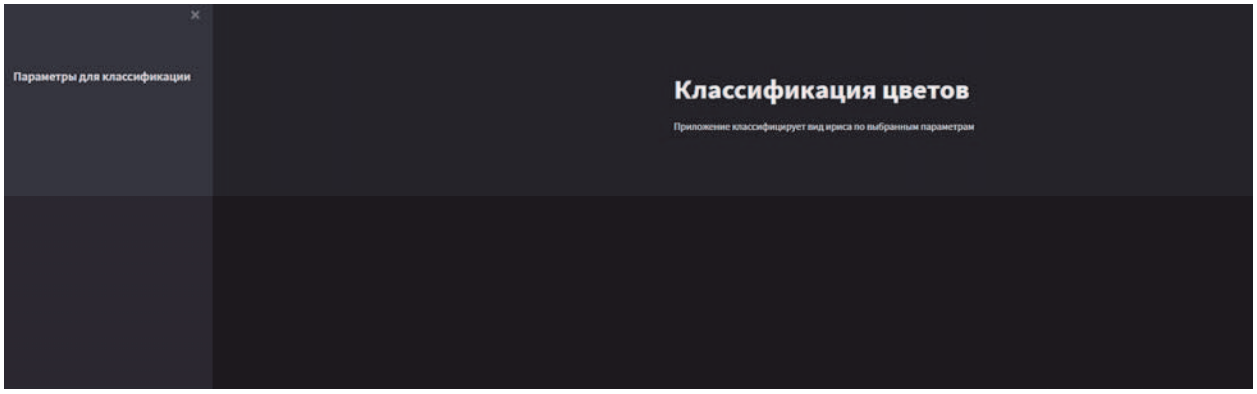


Рисунок 3.1 – Макет классификатора в Streamlit

Далее выведем на экран функционал настройки параметров интересующего ириса для дальнейшей его классификации.

```
def user_input_features():
    sepal_length = st.sidebar.slider('Sepal length', 4.3, 7.9, 5.4)
    sepal_width = st.sidebar.slider('Sepal width', 2.0, 4.4, 3.4)
    petal_length = st.sidebar.slider('Petal length', 1.0, 6.9, 1.3)
    petal_width = st.sidebar.slider('Petal width', 0.1, 2.5, 0.2)
    data = {'sepal_length': sepal_length,
            'sepal_width': sepal_width,
            'petal_length': petal_length,
            'petal_width': petal_width}
    features = pd.DataFrame(data, index=[0])
    return features
df = user_input_features()
```

```
st.subheader('Выбранные параметры пользователем')
st.write(df)

Также построим саму модель классификации ириса:
df = user_input_features()

st.subheader('User Input parameters')
st.write(df)

iris = datasets.load_iris()
X = iris.data
Y = iris.target

clf = RandomForestClassifier()
clf.fit(X, Y)

prediction = clf.predict(df)
prediction_proba = clf.predict_proba(df)
```

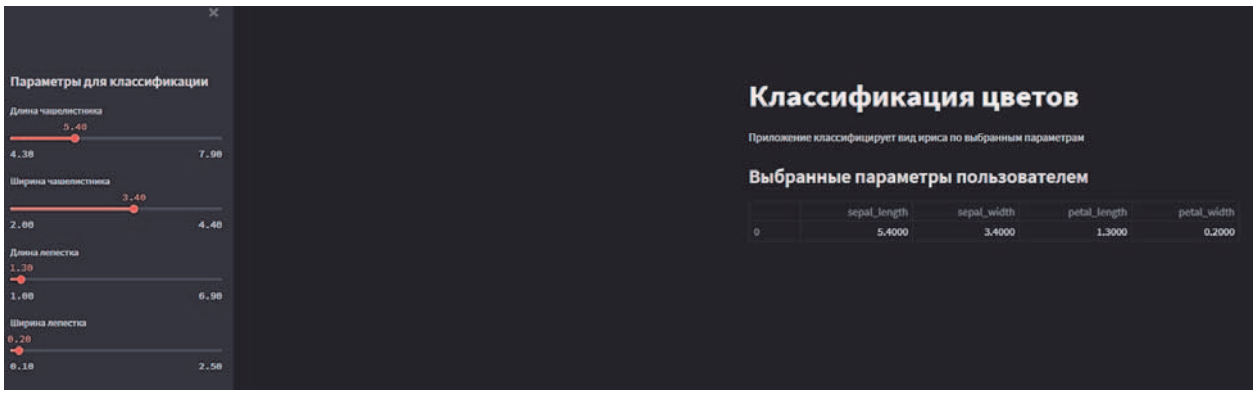


Рисунок 3.2 Отображение выбора параметров цветка

В левой части экрана появилось 4 ползунка, которые позволяют выбрать размеры чашелистика и лепестка исследуемого цветка. Эти параметры сводятся в таблицу по центру экрана.

После добавляется вывод таблицы, в которой пользователь может видеть возможные варианты классификации ириса и индекс видов.

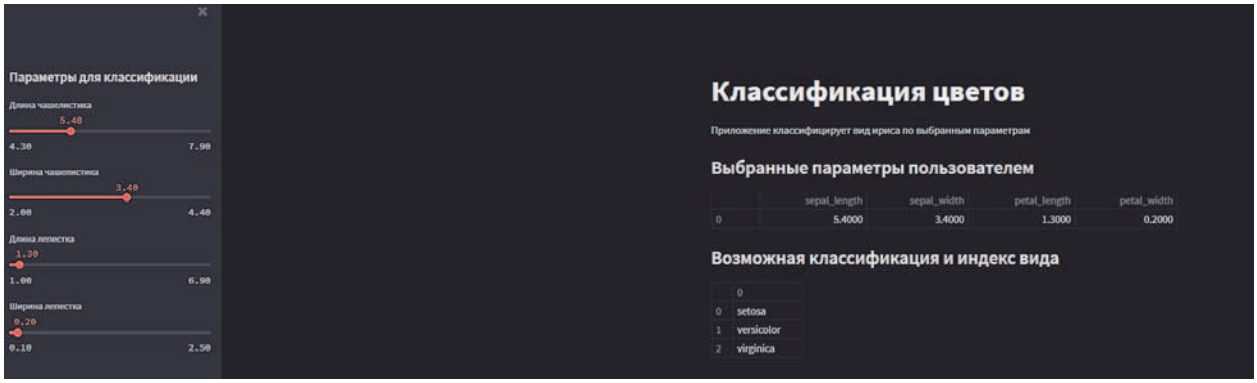


Рисунок 3.3 Возможная классификация ириса

На последнем этапе добавлено прогнозное значение и вероятность прогноза, с помощью которой можно оценить, с какой вероятностью исследуемое растение можно отнести к другим видам.

```
st.subheader('Возможная классификация и индекс вида')
st.write(iris.target_names)

st.subheader('Прогноз')
st.write(iris.target_names[prediction])
#st.write(prediction)

st.subheader('Вероятность прогноза')
st.write(prediction_proba)
```

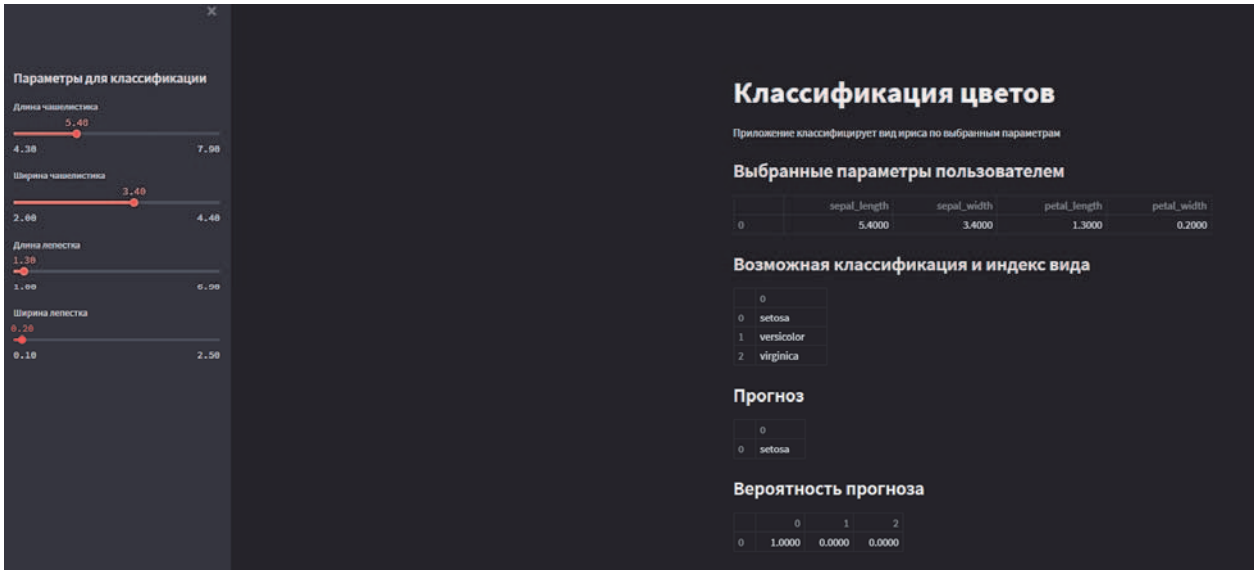


Рисунок 4 – Итоговый вид веб-приложения

Можно видеть, что при выбранных параметрах, алгоритм присваивает ирису вид setosa, причем вероятность правильности этого прогноза равна 1.

Изменяя исходные парамерты, удалось найти, при каких параметрах растения он будет идентифицирован как versicolor с вероятностью 0,99.

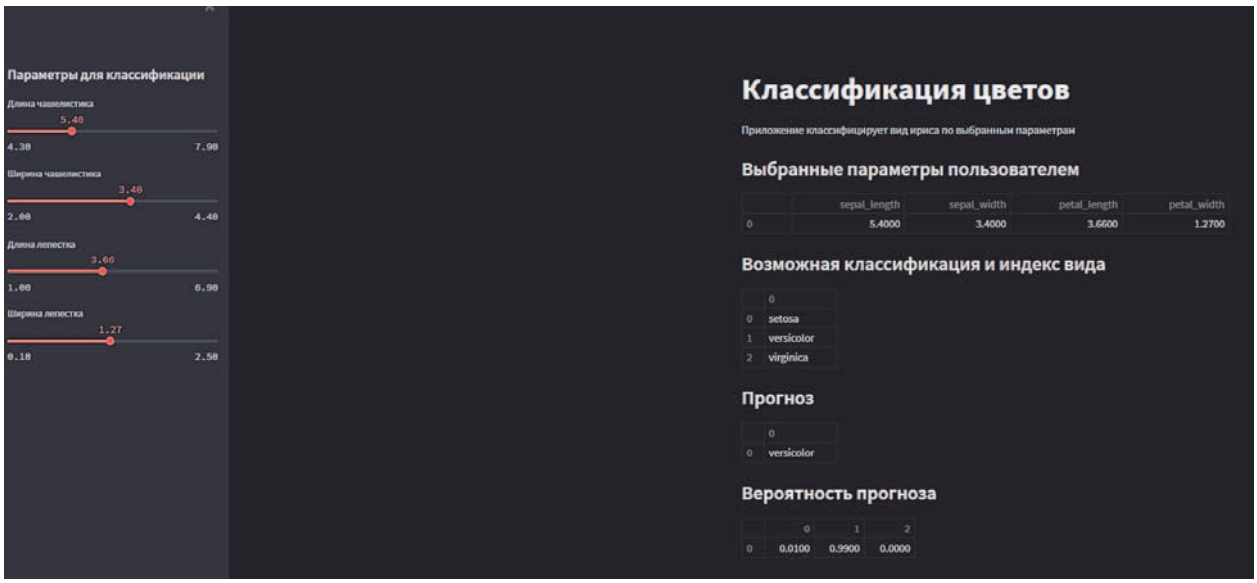


Рисунок 3.5 Идентификация растения как versicolor

Как видим, Streamlit является интуитивно понятным и эффективным инструментом отображения результатов машинного обучения.

Самостоятельная работа

Используя библиотеку Streamlit постройте многостраничное приложение машинного обучения и отобразите результаты любых трех рассмотренных ранее моделей машинного обучения.

Для решения задачи смотри документацию: https://docs.streamlit.io/get-started/tutorials/create-a-multipage-app

Вопросы для самоконтроля

1. В чем заключается задача автоматизации и отображения результатов построения моделей машинного обучения?
2. Основное назначение библиотеки Streamlit?
3. Как запустить приложение Streamlit?
4. Как осуществляется отображение названия столбцов, боковых панелей, таблиц и рисунков в библиотеке Streamlit?
5. Назовите особенности построения многостраничного приложения с применением библиотеки Streamlit.

Лабораторная работа № 3.2
«Построение веб-приложения модели машинного обучения в Gradio»

Теоретические положения

Gradio – модуль Python, позволяющий достаточно быстро создавать веб-приложения машинного обучения (включая API) не имею навыков веб-разработки [1]. Для работы с Gradio должна быть установлена версия Python 3.10 и выше.

Установка Gradio проходит аналогично другим библиотекам Python – через дистрибутив Anaconda: pip install gradio. Установка пакета может быть осуществлена в виртуальной среде.

Для создания простого одностраничного приложения Gradio в Spyder может быть реализован следующий код:

```
import gradio as gr

def greet(name, intensity):
    return "Привет, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"],
)

demo.launch()
```

Рисунок 3.6 Создание одностраничного ML-приложения на Gradio в Spyder

Далее запустив код, мы получаем ссылку на локальный диск, открыв который в веб-браузере мы увидим наше приложение. Приложение также может быть запущено в терминале: run python filename.py.

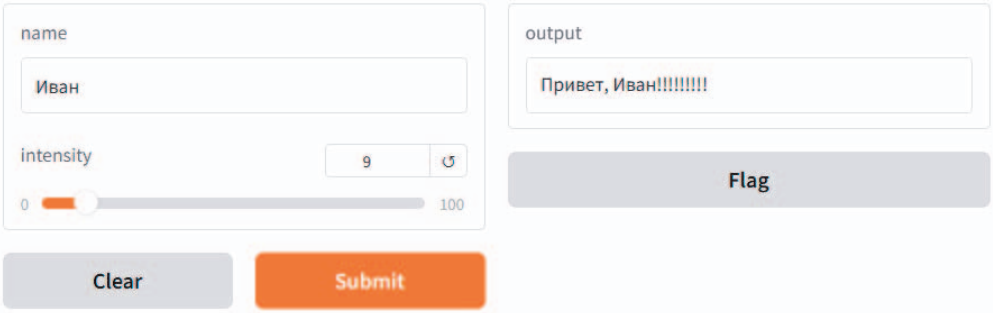


Рисунок 3.7 Фрагмент одностраничного приложения в Gradio

Заполнив имя в строке «name», мы увидим первое приветствие. Как видно из первого рисунка работы для создания первой страницы приложения необходимо было создать класс gr.Interface. Этот класс создается для реализации демо версии приложения машинного обучения. Класс принимает на вход (input) один или более входящих параметров и выдает параметр или более на выходе (output). Данный класс содержит 3 аргумента:

fn: функция для отражения чего-либо в пользовательском интерфейсе;

inputs: компонент Gradio, задаваемый на входе. Количество компонентов на входе должно совпадать с количеством аргументов в функции;

outputs: компонент Gradio для отражения параметров на выходе функции. Количество компонентов должно совпадать с количеством возвращаемых с функцией значений.

Аргумент fn достаточно гибкий и может отображать в графическом интерфейсе любые объекты: простые функции, графики, предсказанные значения и модели машинного обучения.

Gradio включает более 30 компонентов для разработки приложений для машинного обучения, например, gr.Textbox(), gr.Image(), and gr.HTML().

Если ваша функция принимает более одного аргумента, как в примере выше передайте на inputs список входных компонентов, где каждый входной компонент соответствует одному из аргументов функции по порядку. То же самое справедливо и в том случае, если ваша функция возвращает более одного значения: просто передайте список компонентов на outputs. Эта гибкость делает класс Interface очень мощным средством создания демоверсий ML-приложений.

Созданной URL ссылкой можно делиться без применения каких-либо инструментов хостинга. Для этого необходимо указать параметр share=True в функции launch():

demo.launch(share=True) #последняя строка кода в примере выше

После запуска кода вы получите URL-ссылку на ваше приложение, которую можно использовать как стандартную ссылку на какой-либо сайт.

В библиотеке Gradio можно создавать более «кастомизированные» приложения с помощью метода .Blocks(), создавать чат-боты на основе .ChatInterface(), интегрироваться с возможностями JavaScript.

Задание:

Условие. Имеется построенная модель классификации ирисов и библиотека Gradio.

Требуется: отобразить результаты построения модели классификации ирисов при помощи библиотеки Gradio.

Методические указания к выполнению лабораторной работы

Рассмотрим простой пример ML-приложения, которое предсказывает значение одного параметра (целевой переменной y) на основе введенных данных.

Для начала установим библиотеку: pip install gradio

Далее создадим простую модель машинного обучения (линейную регрессию), которая обучается на искусственно сгенерированных данных. Модель должна прогнозировать целевую переменную y под влиянием признаков x1 и x2.

```
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.datasets import make_regression
import gradio as gr

# Генерация синтетических данных
X, y = make_regression(n_samples=1000, n_features=2, noise=20)

# Обучение модели
model = LinearRegression()
model.fit(X, y)
```

Рисунок 3.8 Загрузка библиотек и создание модели регрессии

Затем создадим интерфейс, используя возможности библиотеки Gradio.

```
def predict(x1, x2):
    # Преобразуем входные данные в массив NumPy
    X_new = np.array([[x1, x2]])

    # Делаем прогноз
    prediction = model.predict(X_new)[0]

    return f"Предсказанное значение: {prediction:.2f}"

# Создаем интерфейс
iface = gr.Interface(
    fn=predict,
    inputs=[gr.Number(label="Переменная x1"), gr.Number(label="Переменная x2")],
    outputs=gr.Textbox(),
    title="Простое приложение машинного обучения",
    description="Введите значения переменных x1 и x2, чтобы получить прогноз."
)
# Запускаем приложение
iface.launch(share=True)
```

Рисунок 3.9 Создание ML-приложения для прогнозирования значений на основе линейной регрессии

Обратите внимание, что в методе launch() предусмотрено создание URL: ссылки для того, чтобы можно было поделиться разработанным приложением, например, с будущим пользователем или заказчиком.

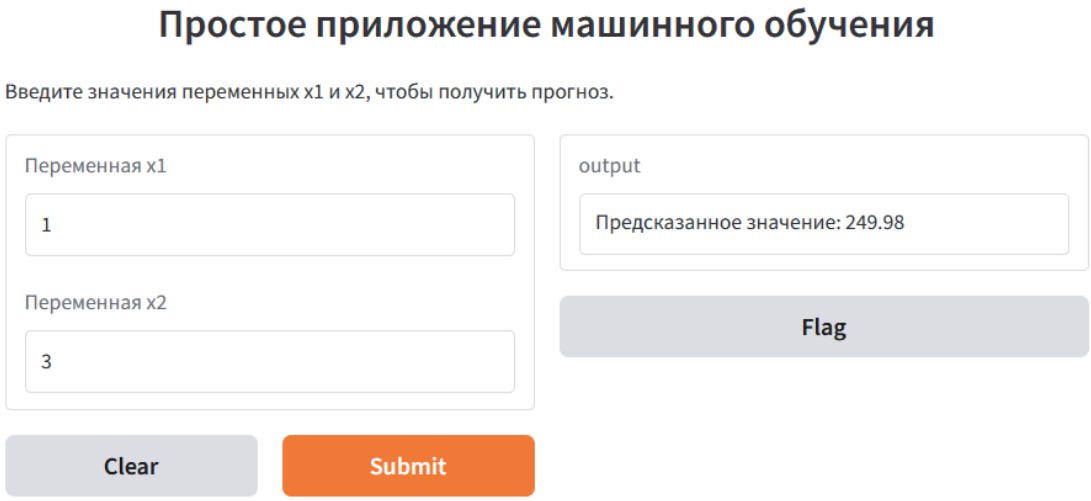


Рисунок 3.10 Фрагмент интерфейса ML-приложения для прогнозирования значений на основе линейной регрессии

На рсиунке видно, что при заданных значениях переменной x1 и x2 прогноз целевой переменной y составил 249,98 каких-то условных единиц.

Самостоятельная работа

Используя библиотеку Gradio постройте многостраничное приложение машинного обучения и отобразите результаты любых трех рассмотренных ранее моделей машинного обучения.

Для решения задачи и более подробного изучения возможности приложения смотри документацию: https://www.gradio.app/ .

Вопросы для самоконтроля

1. В чем на ваш взгляд состоит основное отличие библиотек Gradio и Streamlit?
2. Перечислите основные методы, которые могут быть реализованы в Gradio?
3. Как создать URL-ссылку для приложения машинного обучения на adio?
4. Какой метод отвечает за более «кастомизированную» разработку на Gradio?
5. Какой метод используется для создания чат-бота на Gradio?
6. Какую функцию необходимо использовать для отражения результатов прогноза или иных объектов в интерфейсе Gradio?
7. Какие есть способы запуска приложения Gradio?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Gradio. Официальный сайт библиотеки Python Gradio. URL: https://www.gradio.app/ (дата обращения: 25.11. 2024).

2. Installing scikit-learn. URL: https://scikit-learn.org/stable/install.html (Дата обращения: 28.09.2024).

3. KNeighborsClassifier. URL: https://scikit-learn.org/stable/modules /generated/sklearn.neighbors.KNeighborsClassifier.html (Дата обращения: 28.09.2024).

4. LazyPredict Documentation. URL: https://lazypredict.readthedocs.io/ en/latest/usage.html (Дата обращения: 27.09.2024).

5. Logit vs Probit Models: Differences, Examples. URL: https://vitalflux.com/logit-vs-probit-models-differences-examples/ (Дата обращения: 26.09.2024).

6. Simple and Multiple Linear Regression in Python. Available online: https://towardsdatascience.com/simple-and-multiple-linear-regression-in-python-c928425168f9 (Дата обращения: 26.09.2024).

7. Streamlit. Официальный сайт библиотеки Python streamlit. URL: https://streamlit.io/(дата обращения: 23.11. 2024).

8. TfidfVectorizer. URL: https://scikit-learn.org/stable/modules/ generated/sklearn.feature_extraction.text.TfidfVectorizer.html (Дата обращения: 28.09.2024).

9. Train Multiple ML Models using Lazypredict in Python – 2024. URL: https://machinelearningprojects.net/train-models-with-lazypredict/ (Дата обращения: 27.09.2024).

10. Мультиагентное обучение с подкреплением : учебное пособие / Алфимцев А. Н. ; МГТУ им. Н. Э. Баумана, 2-е изд., испр. - М. : Изд-во МГТУ им. Н. Э. Баумана, 2022. - 222 с.

11. Пример решения задачи множественной регрессии с помощью Python. [Электронный ресурс]. - Режим доступа: https://habr.com/ru/ post/206306/ (Дата обращения: 26.09.2024).

12. Системы искусственного интеллекта : учебник / Девятков В. В. - М. : Изд-во МГТУ им. Н. Э. Баумана, 2023. - 280 с.

13. Тринадцать способов настроить визуализацию матрицы корреляции. [Электронный ресурс]. - Режим доступа: https://datastart.ru/ blog/read/seaborn-heatmaps-13-sposobov-nastroit-vizualizaciyu-matricy-korrelyacii (Дата обращения: 30.09.2024).

14. Тюрин, А.Г. Кластерный анализ, методы и алгоритмы кластеризации / А.Г. Тюрин // Вестник МГТУ МИРЭА. – 2014. – №2. с. 86-9.

15. Учебник по Python. Создание модели для классификации клиентов с использованием машинного обучения SQL. [Электронный ресурс]. - Режим доступа: https://docs.microsoft.com/ru-ru/sql/machine-learning/tutorials/python-clustering-model-build?view=sql-server-ver15 (Дата обращения: 23.09.2024).

Учебное издание

Демичев Вадим Владимирович
Быков Денис Витальевич
Невзоров Александр Сергеевич
Токарев Виктор Сергеевич

БОЛЬШИЕ ДАННЫЕ

Учебное пособие

Сдано в набор 05.12.2024. Подп. в печ. 12.12.2024.
Формат 60×88/16. Бумага офсетная.
Усл.печ.л. 5,5 Тираж 500 экз.

ISBN 978-5-907933-16-3

Издательство Научный консультант
123007, г. Москва, Хорошевское ш., 35к2, офис 508.
Тел.: +7 (926) 609-32-93, +7 (499) 195-60-77 www.n-ko.ru keyneslab@gmail.com