МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Национальный исследовательский университет «МЭИ»

Институт ИВТИ

Кафедра ПМИИ

Дисциплина: «Программная инженерия»

Отчет по курсовой работе на тему:

«Анализ датасетов Iris Dataset и Boston Housing Dataset»

Выполнил: студент группы А-13а-19

Рамазанов Н. М.

Научный руководитель: доц. каф. ПМИИ

Кружилов И. С.

Москва, 2022

Оглавление

1. Введение.

В современном мире каждый день в том или ином виде накапливаются терабайты информации. Данные о посетителях сайта, о заказах через интернет-магазин, о подписавшихся на рассылку пользователях – все это примеры больших объемов данных. Зачем их собирают?

В большинстве случаев это делается в коммерческих целях – ради извлечения какой-либо выгоды. Например, сбор данных о пользователях помогает сайтам определить, что тому или иному человеку наиболее интересно, и основываясь на этом показать самые подходящие рекламные объявления; тем самым заказчики могут получать большую прибыль за покупки рекламируемого продукта, а владельцы сайтов – за приобретение рекламы. Но выделяют и другие причины:

- персонализация показываемого контента;
- защита конфиденциальных данных;
- предотвращение публикации незаконных материалов;
- проведение аналитических исследований.

Таким образом, на сегодняшний день информация играет крайне важную роль во всех сферах жизни. Однако, в большинстве своем собираемые данные находятся в неупорядоченном виде, и сами по себе не несут никакой практической пользы. Для того чтобы на их основе можно было делать какие-то выводы, которые могут быть необходимы для решения конкретной проблемы или задачи, с ними проводят серьёзную работу. Она заключается в первичной обработке, исследовании, фильтрации, визуализации данных, построении моделей, формулировании гипотез и их проверке и, в конечном счёте, получении некоторых знаний. Все перечисленные методы объединяет процесс, получивший название «*анализ данных*».

2. Необходимый теоретический материал.

Дадим строгое определение анализа данных.

Анализ данных (далее – АД) – это область науки на стыке математики и информатики, изучающая алгоритмы и методы извлечения из массивов данных полезных знаний, а также непосредственно сам процесс обработки, исследования, интерпретации данных с целью получения результатов, которые могут помочь принять решение по тому или иному вопросу.

В данной работе будет рассматриваться и реализовываться его разновидность – интеллектуальный анализ данных (*data mining*, далее – ИАД). Наиболее точное его определение дал Григорий Пятецкий-Шапиро, который и ввел это понятие в 1989 году – это процесс обнаружения в сырых данных ранее неизвестных, нетривиальных, практически полезных и доступных интерпретации знаний, необходимых для принятия решений в различных сферах человеческой деятельности. Его можно сравнить с добычей полезных ископаемых – подобно геологу, ищущему месторождения нефти в толщах горных пород, аналитик ищет скрытые знания в таблицах и больших базах данных.

Задачи, решаемые с помощью ИАД, в свою очередь подразделяются на описательные, в которых требуется описать общие свойства данных, и предсказательные, в которых требуется проанализировать данные и на основе полученных выводов сделать прогноз. В данной курсовой работе будут реализованы решения предсказательных задач, а именно:

– классификация объектов по заранее определённым классам: предсказание вида ириса (щетинистый, разноцветный, виргинский) по его параметрам (длина и ширина чашелистика, длина и ширина лепестков);

– регрессионный анализ: предсказание стоимости частных домов в Бостоне.

В процессе ИАД выделяют следующие основные этапы:

1. Изучение предметной области и постановка цели и задач ИАД;
2. Сбор данных;
3. Подготовка данных: очистка, преобразование и т. д.;
4. Анализ данных, включающий в себя:
    a. Выбор модели (алгоритма АД);
    b. Подбор оптимальных параметров и способа её обучения;
    c. Обучение модели;
    d. Проверка качества обучения;
5. Анализ и интерпретация полученных результатов;
6. Использование новых знаний.

Одним из основных инструментов аналитика является язык программирования Python и его библиотеки – pandas, numpy, matplotlib, seaborn, sklearn и другие. Приведем краткую информацию об использованных в работе библиотеках и их функциях.

2.1. Библиотека pandas.

Pandas – это библиотека с открытым исходным кодом, предоставляющая высокопроизводительные, простые в использовании структуры данных и инструменты анализа данных для языка программирования Python.

Использованные в работе структуры данных и методы:

- *pandas.DataFrame* – основная структура данных pandas («датафрейм»), представляет собой двумерные, изменяемые, потенциально неоднородные табличные данные;
- *pandas.DataFrame.info* – метод, который выводит информацию о датафрейме, включающую в себя типы данных индексов и столбцов, количество ненулевых значений, а также объем используемой памяти;

- *pandas.DataFrame.set_axis* – изменяет значения в заданной оси;
- *pandas.DataFrame.reset_index* – сбрасывает индексный столбец на индекс по умолчанию;
- *pandas.DataFrame.head* – выводит на экран первые *n* строк датафрейма;
- *pandas.DataFrame.apply* – применяет функцию к заданной оси датафрейма;
- *pandas.DataFrame.corr* – вычисляет попарную корреляцию столбцов, исключая пропуски в данных;
- *pandas.DataFrame.round* – округляет числовые значения в датафрейме до заданного числа десятичных знаков;
- *pandas.DataFrame.columns* – названия столбцов датафрейма;
- *pandas.DataFrame.describe* – рассчитывает описательные статистики, такие как: количество элементов, среднее значение, стандартное отклонение, минимум, максимум и значения, отделяющие 25, 50 и 75% данных.

2.2. Библиотека numpy.

NumPy – это фундаментальный пакет для научных вычислений на Python. Это библиотека Python, которая предоставляет такой объект, как многомерный массив, различные производные объекты и набор процедур для быстрых операций с массивами, включая математические, логические, манипуляции с формой, сортировку, выбор, ввод/вывод, дискретные преобразования Фурье, базовую линейную алгебру, базовые статистические операции, случайное моделирование и многое другое.

Использованные в работе структуры данных и методы:

- *numpy.ndarray* – основная структура данных numpy, представляет собой многомерный однородный массив элементов фиксированного размера;
- *numpy.sqrt* – возвращает квадратный корень заданного значения;
- *numpy.corrcoef* – возвращает коэффициенты корреляции Пирсона для заданных аргументов;
- *numpy.triu* – возвращает копию массива, в которой элементы ниже k-ой диагонали являются нулями;
- *numpy.ones_like* - возвращает массив единиц той же формы и типа, что и исходный массив;
- *numpy.ndarray.mean* - возвращает среднее значение элементов массива вдоль заданной оси.

2.3. Библиотека matplotlib.

Matplotlib – это комплексная библиотека для создания статических, анимированных и интерактивных визуализаций на Python.

В работе использовался только модуль *matplotlib.pyplot* – он представляет собой интерфейс для matplotlib, обеспечивает неявный способ построения графиков, также выводит их на экран и действует как графический менеджер рисунков.

Использованные в работе методы:

- *matplotlib.pyplot.show* – отображает все созданные графики на одном рисунке;
- *matplotlib.pyplot.figure* – создает новый рисунок или активирует существующий;
- *matplotlib.pyplot.subplot* – добавляет оси к существующему рисунку или извлекает существующие;

- *matplotlib.pyplot.scatter* – точечная диаграмма зависимости *y* от *x* с различным размером и / или цветом маркера;
- *matplotlib.pyplot.title* – добавляет заголовок к заданной оси;
- *matplotlib.pyplot.xlabel* – добавляет подпись к оси абсцисс графика;
- *matplotlib.pyplot.ylabel* – добавляет подпись к оси ординат графика.

2.4. Библиотека seaborn.

Seaborn – это библиотека визуализации данных Python, основанная на matplotlib. Она обеспечивает высокоуровневый интерфейс для рисования привлекательной и информативной статистической графики.

Использованные в работе методы:

- *seaborn.pairplot* – строит графики попарных зависимостей столбцов в наборе данных;
- *seaborn.heatmap* – строит тепловую карту для прямоугольных данных;
- *seaborn.displot* – строит многосегментные гистограммы распределения данных нескольких подмножеств, также подходит для построения единичной гистограммы.

2.5. Библиотека sklearn.

Scikit-learn (sklearn) – это библиотека с открытым исходным кодом, основанная на возможностях NumPy и SciPy, предоставляющая широкий спектр возможностей и алгоритмов анализа данных и машинного обучения.

Использованные в работе пакеты:

- *sklearn.datasets* – пакет, в который встроено несколько небольших датасетов, подходящих для знакомства с анализом данных, которые можно загрузить напрямую в программе, не прибегая к сторонним источникам;

- *sklearn.linear_model* – предоставляет набор моделей, в которых целевое значение, как ожидается, будет линейной комбинацией признаков;
- *sklearn.model_selection* – предоставляет различные методы для поиска оптимальных параметров модели и увеличения точности предсказаний модели;
- *sklearn.metrics* – включает в себя функции оценки, показатели производительности, попарные показатели и вычисления разниц для проверки качества модели.

Использованные в работе методы:

- *sklearn.datasets.load_iris* – загружает датасет «Ирисы Фишера»;
- *sklearn.datasets.load_boston* – загружает датасет «Цены на жилье в Бостоне»;
- *sklearn.model_selection.train_test_split* – выполняет случайное разделение набора данных на обучающую (на которой модель будет обучаться) и тестовую (на которой модель будет тестироваться) выборки;
- *sklearn.model_selection.cross_validate* – выполняет кросс-валидацию (перекрёстную проверку), то есть производит случайное разбиение на обучающую и тестовую выборки, а затем вычисляет заданную метрику качества заданное число раз, чтобы повысить надежность модели;
- *sklearn.model_selection.GridSearchCV* – производит поиск оптимальных параметров по заданным значениям, при которых заданная метрика качества модели будет наилучшей;
- *sklearn.linear_model.SGDClassifier* – реализует регуляризованные (то есть с определенными ограничениями, чтобы сделать обучение проще и быстрее, а также предотвратить переобучение) линейные

модели со стохастическим градиентным спуском (метод оптимизации некоторой функции с помощью оценки реального градиента функции градиентом, вычисленным из случайно выбранного подмножества данных);

- *sklearn.linear_model.SGDRegressor* – реализует модель линейной регрессии с оптимизацией стохастическим градиентным спуском;
- *sklearn.linear_model.LinearRegression* – обычная модель линейной регрессии по методу наименьших квадратов;
- *sklearn.ensemble.RandomForestRegressor* – модель регрессии на основе алгоритма «случайного леса»: используется ансамбль деревьев решений, каждое из которых по отдельности имеет низкую точность прогноза, но в совокупности дают существенно более высокий результат;
- *sklearn.linear_model.*.fit* – тренирует модель на заданном обучающем наборе (где * - выбранная модель для обучения);
- *sklearn.linear_model.*.predict* – возвращает предсказанные моделью метки классов (в случае классификации) или числовые значения целевой переменной (в случае регрессионного анализа) на заданном тестовом наборе;
- *sklearn.metrics.make_scorer* – создает метрику качества модели на основе показателя производительности или функции потерь;
- *sklearn.metrics.classification_report* – создаёт текстовый отчет с основными метриками качества классификации:
    1. *accuracy* – доля верных предсказаний модели (как принадлежность объекта к классу, так и его непринадлежность).
    2. *precision* – это метрика, показывающая, какую долю среди всех объектов, причисленных классификатором к

некоторому классу, составляют объекты, действительно являющиеся членами этого класса.

3. *recall* – показывает, какую долю среди всех объектов, действительно являющихся членами некоторого класса, классификатор причислил к этому классу.
4. *f1-score* – это метрика, объединяющая *presicion* и *recall*: берется среднее гармоническое от этих метрик, умноженное на 2 (чтобы в случае равенства их единице, *f1-score* тоже равнялось единице).

- *sklearn.metrics.accuracy_score* – возвращает значение точности классификации;
- *sklearn.metrics.mean_squared_error* – возвращает значение среднеквадратичной ошибки модели регрессии;
- *sklearn.metrics.r2_score* – возвращает коэффициент детерминации (показывает, какая доля дисперсии целевой переменной обусловлена влиянием факторов, учтенных моделью).

3. Анализ задания.

Рассмотрим общую информацию по каждому из двух датасетов, а также сформулируем постановки задач классификации объектов и регрессионного анализа.

3.1. Датасет «Ирисы Фишера».

Датасет «Ирисы Фишера» состоит из 150-ти строк, каждая из которых содержит информацию о 3-х видах ириса: *“Setosa”* – ирис щетинистый; *“Versicolor”* – ирис разноцветный; *“Virginica”* – ирис виргинский. Каждый класс представлен в наборе данных 50 экземплярами. Для каждого экземпляра измерялись четыре характеристики (в сантиметрах):

- *sepal_length* – длина чашелистика;

- *sepal_width* – ширина чашелистика;
- *petal_length* – длина лепестков;
- *petal_width* – ширина лепестков.

Целевая переменная - номер класса, соответствующего тому или иному виду ириса: 0 – *“Setosa”*, 1 – *“Virginica”*, 2 – *“Versicolor”*.

Постановка задачи: необходимо построить и обучить модель предсказывать принадлежность объекта к некоторому классу по заданным параметрам (иначе говоря, классифицировать экземпляры ириса по видам, имея данные о размерах его чашелистика и лепестков).

3.2. Датасет «Цены на жилье в Бостоне».

Датасет «Цены на жилье в Бостоне» состоит из 506 строк, каждая из которых содержит информацию об одном экземпляре недвижимости в городе Бостон. Каждый экземпляр в наборе данных имеет 13 различных характеристик (атрибутов):

- CRIM – уровень преступности на душу населения;
- ZN – доля земли под жилую застройку, выделенной для участков площадью более 25 000 кв. футов;
- INDUS – доля акров, не относящихся к розничной торговле, на город;
- CHAS – переменная реки Чарльз (=1, если участок рядом с рекой; 0 – в противном случае);
- NOX – концентрация оксидов азота (частей на 10 миллионов);
- RM – среднее количество комнат в доме;
- AGE – доля частных домов, построенных до 1940 года;
- DIS – взвешенные расстояния до пяти бостонских центров занятости;
- RAD – индекс доступности ближайших автомагистралей;

- TAX – полная ставка налога на имущество на 10 тыс. долл. США;
- PTRATIO – соотношение учеников и учителей в городе;
- B – величина, равная $1000 \cdot (B_k - 0{,}63)^2$, где B_k – доля чернокожего населения в городе.
- LSTAT – процент населения с более низким социальным статусом;

Целевая переменная – MEDV – средняя стоимость частных домов (в тыс. долл.).

Постановка задачи: необходимо построить и обучить модель прогнозировать цену на жилье по заданным входным параметрам.

4. Реализация и тестирование.

Опишем работу с датасетами, построение и обучение моделей, а также тестирование их качества.

Вначале необходимо импортировать библиотеки, которые понадобятся для выполнения работы: библиотека *numpy* – для работы с массивами, матрицами и упрощения выполнения различных математических операций; *pandas* – для удобства работы с табличными данными; *matplotlib* и *seaborn* – для визуализации промежуточных и конечных результатов исследования; *sklearn* – содержит комплексные средства проведения анализа данных непосредственно для решения поставленных задач (импортируем не всю библиотеку, а только необходимые классы и методы).

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import datasets
from sklearn import linear_model
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn import metrics
```

Рис. 1. Импорт необходимых библиотек.

Дальнейшие действия будем описывать отдельно для каждого из двух датасетов.

4.1. Датасет «Ирисы Фишера».

Загрузка и предобработка данных.

Загрузим датасет из библиотеки *sklearn* и выведем обзорную информацию о том, какие данные в нем содержатся.

```
iris = datasets.load_iris()
print(iris.feature_names)
print(iris.data[:5])
print(iris.target_names)
print(iris.target)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[[5.1 3.5 1.4 0.2]
 [4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
['setosa' 'versicolor' 'virginica']
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Рис. 2 (а). Обзор данных в датасете «Ирисы Фишера».

Преобразуем загруженные данные в датафрейм для удобства работы. Названия столбцов приведем в единый стиль – т. н. «змеиный регистр» (пробелы заменяются знаком нижнего подчеркивания, все символы переводятся в нижний регистр). Добавим в таблицу столбец *iris_category*, обозначающий числовое обозначение вида ириса, и столбец *iris_name* – название вида ириса.

```
iris_data = pd.DataFrame(data=iris.data, columns=iris.feature_names)
iris_data.set_axis(['sepal_length_cm', 'sepal_width_cm',
                    'petal_length_cm', 'petal_width_cm'], axis='columns', inplace=True)
iris_data['iris_category'] = iris.target
iris_data['iris_name'] = iris_data['iris_category'].apply(lambda x: iris.target_names[x])
iris_data.head()
```

	sepal_length_cm	sepal_width_cm	petal_length_cm	petal_width_cm	iris_category	iris_name
0	5.1	3.5	1.4	0.2	0	setosa
1	4.9	3.0	1.4	0.2	0	setosa
2	4.7	3.2	1.3	0.2	0	setosa
3	4.6	3.1	1.5	0.2	0	setosa
4	5.0	3.6	1.4	0.2	0	setosa

Рис. 3 (а). Подготовка данных.

Просмотрим общую информацию о данных с помощью метода *info*.

```
iris_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   sepal_length_cm  150 non-null    float64
 1   sepal_width_cm   150 non-null    float64
 2   petal_length_cm  150 non-null    float64
 3   petal_width_cm   150 non-null    float64
 4   iris_category    150 non-null    int32
 5   iris_name        150 non-null    object
dtypes: float64(4), int32(1), object(1)
memory usage: 6.6+ KB
```

Рис. 4 (а). Общая информация о данных.

В таблице нет пропущенных значений, типы данных в столбцах заданы верно. В реальных условиях такое случается редко. Но поскольку для анализа взят готовый датасет из библиотеки *sklearn*, в нем уже все подготовлено для дальнейшей работы, проводить предобработку нет необходимости.

Визуализация данных и нахождение зависимостей.

Построим матрицу точечных графиков зависимостей параметров ириса друг от друга, отобразив виды ириса точками различных цветов.

```
sns.pairplot(iris_data[iris_data.columns[iris_data.columns != 'iris_category']],
             hue="iris_name", markers=["o", "s", "D"], palette="bright")
plt.show()
```

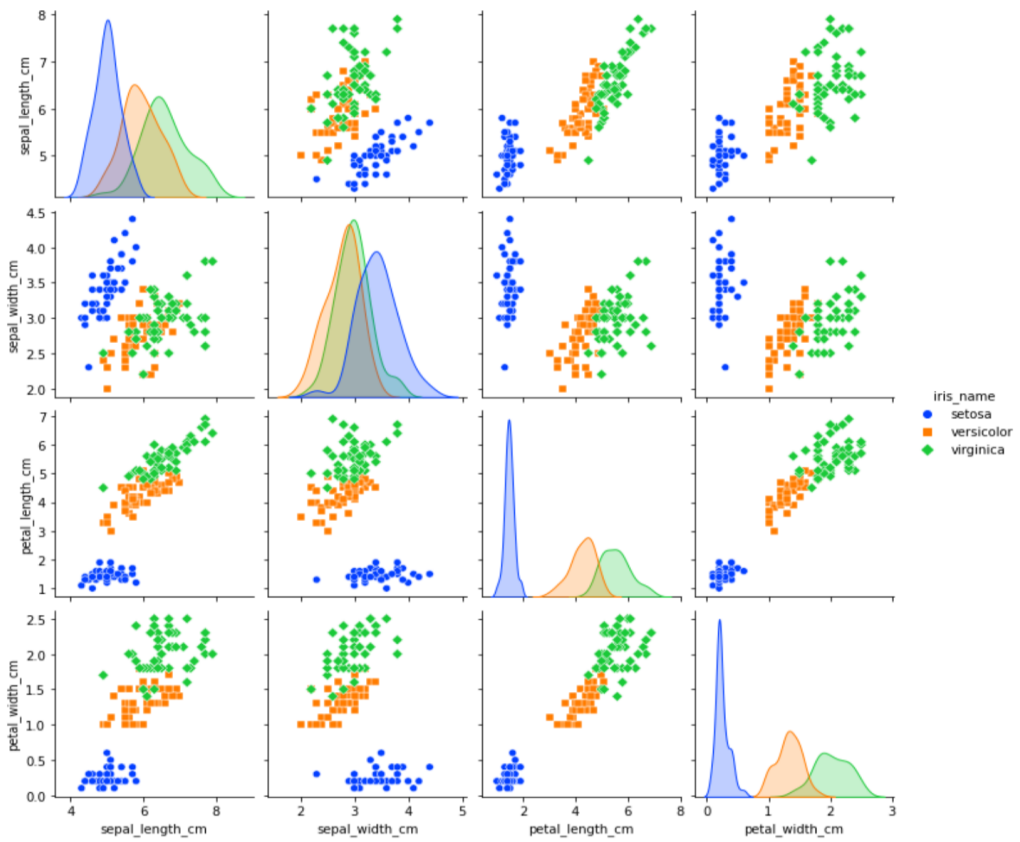


Рис. 5 (а). Графики зависимостей числовых параметров.

Изучив графики распределения значений, можем прийти к следующим выводам:

- длина и ширина лепестков сильно между собой коррелируют – множества точек на графике вытянуты вдоль одной линии, что говорит о наличии линейной зависимости этих параметров друг от друга;
- у ирисов вида *setosa* множество значений любого параметра не пересекается с множествами значений других двух видов – это значит, что он легко линейно отделим от них.

Докажем первую гипотезу строго математически, вычислив коэффициент корреляции этих параметров.

```
np.corrcoef(iris_data['petal_length_cm'], iris_data['petal_width_cm'])[0, 1]
```

```
0.9628654314027961
```

Рис. 6 (а). Вычисление коэффициента корреляции.

Коэффициент действительно очень высокий – 0.96 – практически полная линейная зависимость. Теперь построим тепловую карту коэффициентов корреляции попарно для всех переменных.

```
corr = iris_data[iris_data.columns[iris_data.columns != 'iris_category']].corr()
mask = np.triu(np.ones_like(corr, dtype=np.bool))
sns.heatmap(corr, mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG').set_title(
                            'Тепловая карта корреляции', fontdict={'fontsize':12});
```

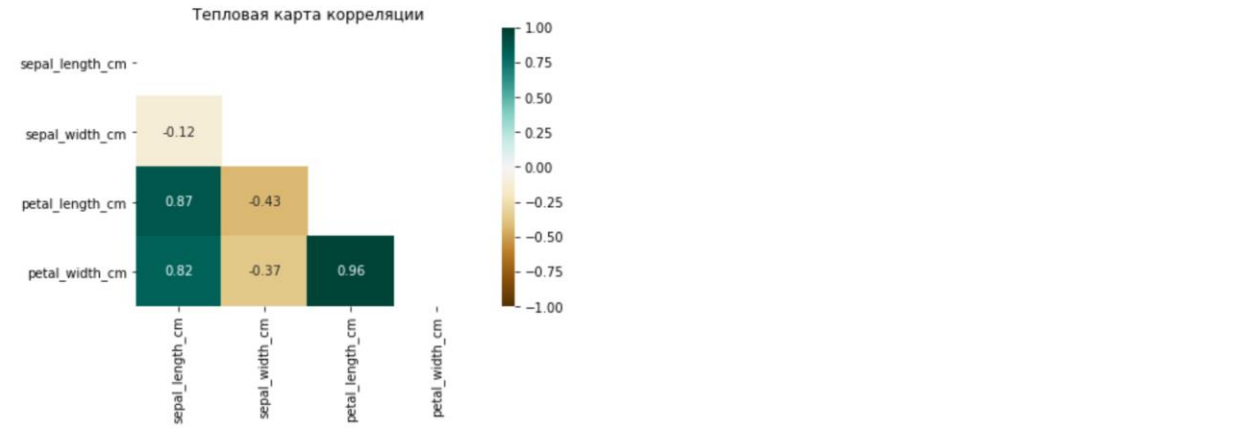


Рис. 7 (а). Тепловая карта корреляции.

Заметим, что помимо того, что длина и ширина лепестков очень хорошо коррелируют между собой, они также неплохо коррелируют с длиной

чашелистика - коэффициенты равны 0.87 и 0.82 соответственно. Таким образом, если мы будем знать длину лепестков, мы сможем с приличной точностью предсказать их ширину, и наоборот. А зная длину чашелистика - сможем с неплохой точностью (но меньше, чем в первом случае) предугадать длину и ширину лепестков, и наоборот.

Построение и обучение модели.

Выберем для нашей задачи линейную модель с общеизвестным методом стохастического градиентного спуска для оптимизации обучения. Сначала разделим данные на обучающую и тестовую выборку (сделаем это в отношении 70:30, указав параметр *test_size=0.3*). Далее объявим классификатор, указав только параметр *random_state=42*, чтобы при каждом запуске кода получать одинаковый результат, обучим модель при помощи метода *fit* и выведем метрики качества модели: *accuracy*, *precision*, *recall* и *f1-score* при помощи функции *classification_report*.

```
train_data, test_data, train_labels, test_labels = train_test_split(
    iris_data.drop(columns=['petal_length_cm', 'iris_category', 'iris_name']),
    iris_data['iris_name'], test_size=0.3, random_state=42)
model = linear_model.SGDClassifier(random_state=42)
model.fit(train_data, train_labels)
model_predictions = model.predict(test_data)
print(metrics.classification_report(test_labels, model_predictions))
```

```
              precision    recall  f1-score   support

      setosa       0.58      1.00      0.73        19
  versicolor       0.00      0.00      0.00        13
   virginica       0.92      0.85      0.88        13

    accuracy                           0.67        45
   macro avg       0.50      0.62      0.54        45
weighted avg       0.51      0.67      0.56        45
```

Рис. 8 (а). Построение и обучение модели, вывод метрик качества.

Точность предсказания *accuracy* получается неплохая, модель выдала порядка 67% верных предсказаний, однако, что мы видим? Модель не отнесла ни одного экземпляра ириса к виду *versicolor*! Это значит, что наша модель нуждается в доработке.

Одним из распространенных методов улучшения качества классификатора является подбор наиболее оптимальных параметров. Для этого служит метод *GridSearchCV*.

Рассмотрим параметры *SGDClassifier*, установленные по умолчанию:

*class sklearn.linear_model.SGDClassifier(loss='hinge', *, penalty='l2', alpha=0.0001, l1_ratio=0.15, fit_intercept=True, max_iter=1000, tol=0.001, shuffle=True, verbose=0, epsilon=0.1, n_jobs=None, random_state=None, learning_rate='optimal', eta0=0.0, power_t=0.5, early_stopping=False, validation_fraction=0.1, n_iter_no_change=5, class_weight=None, warm_start=False, average=False)*

Подберем оптимальное сочетание параметров *loss* – функция потерь, *alpha* – множитель регуляризации и *max_iter* – количество проходов по обучающим данным.

```
parameters = {
    'loss': ['hinge', 'log'],
    'alpha': np.linspace(1e-6, 0.1, 6),
    'max_iter': np.arange(100, 1000, 10)
}
grid = GridSearchCV(model, parameters, cv=10)
grid.fit(train_data, train_labels)
print(grid.best_params_)
print(grid.best_estimator_)
```

```
{'alpha': 0.020000800000000003, 'loss': 'log', 'max_iter': 100}
SGDClassifier(alpha=0.020000800000000003, loss='log', max_iter=100,
              random_state=42)
```

Рис. 9 (а). Нахождение оптимальных параметров.

Теперь получим предсказания от этой модели и выведем метрики.

```
grid_pred = grid.predict(test_data)
print(metrics.classification_report(test_labels, grid_pred))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      0.46      0.63        13
   virginica       0.65      1.00      0.79        13

    accuracy                           0.84        45
   macro avg       0.88      0.82      0.81        45
weighted avg       0.90      0.84      0.83        45
```

Рис. 10 (а). Метрики качества модели с оптимальными параметрами.

Точность предсказания *accuracy* получается очень хорошая, модель выдала порядка 84% верных предсказаний, идеально отличает вид *setosa*, но довольно часто путает *versicolor* и *virginica*. Что можно сделать, чтобы улучшить качество модели в этом аспекте?

Немного ранее из рассмотрения модели был исключен столбец *petal_length_cm*, поскольку он имеет высокую корреляцию с параметром

petal_width_cm. Что, если вместо этого создать единый признак - комбинацию этих двух параметров? Запишем в новый столбец произведение длины и ширины лепестка, таким образом, новый признак - площадь лепестка. Затем снова обучим модель, получим предсказания и выведем метрики качества.

```
iris_data['petal_area'] = iris_data['petal_length_cm'] * iris_data['petal_width_cm']
train_data, test_data, train_labels, test_labels = train_test_split(
    iris_data[['sepal_length_cm', 'sepal_width_cm', 'petal_area']],
    iris_data['iris_name'], test_size=0.3, random_state=42)
model = linear_model.SGDClassifier(loss='log', alpha=0.02, max_iter=100, random_state=42)
model.fit(train_data, train_labels)
model_predictions = model.predict(test_data)
print(metrics.classification_report(test_labels, model_predictions))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      0.85      0.92        13
   virginica       0.87      1.00      0.93        13

    accuracy                           0.96        45
   macro avg       0.96      0.95      0.95        45
weighted avg       0.96      0.96      0.96        45
```

Рис. 11 (а). Метрики качества модели с новым признаком.

Получили очень высокие показатели всех метрик качества. Осталась одна деталь, которая до этого оставалась такой, какой была взята в самом начале. Изначально был выбран размер тестовой выборки – 30% от общего числа данных. Но если попробовать изменять ее размер, точность и остальные метрики могут как улучшаться, так и ухудшаться. Для получения результатов при различных разбиениях и, тем самым, повышения надежности модели служит метод cross-validation, или перекрестной проверки.

```
real_class = []
pred_class = []

def report(real_labels, pred_labels):
    real_class.extend(real_labels)
    pred_class.extend(pred_labels)
    return metrics.accuracy_score(real_labels, pred_labels)

cross_val = cross_validate(model, test_data, test_labels, cv=10,
                           scoring=metrics.make_scorer(report))
print(metrics.classification_report(real_class, pred_class))
```

```
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        19
  versicolor       1.00      0.92      0.96        13
   virginica       0.93      1.00      0.96        13

    accuracy                           0.98        45
   macro avg       0.98      0.97      0.97        45
weighted avg       0.98      0.98      0.98        45
```

Рис. 12 (а). Метрики качества модели после кросс-валидации.

Получили еще более высокие результаты, что может говорить об очень хорошем качестве построенной и обученной модели.

4.2. Датасет «Цены на жилье в Бостоне».

Загрузка и предобработка данных.

Проделаем аналогичные действия - загрузим датасет из библиотеки *sklearn* и трансформируем его в датафрейм. Добавим также столбец с целевыми данными - средней стоимостью частных домов (в тыс. долл.).

```
boston = datasets.load_boston()
print(boston.feature_names)
print(boston.data[:5])
print(boston.target[:5])
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
[[6.3200e-03 1.8000e+01 2.3100e+00 0.0000e+00 5.3800e-01 6.5750e+00
  6.5200e+01 4.0900e+00 1.0000e+00 2.9600e+02 1.5300e+01 3.9690e+02
  4.9800e+00]
 [2.7310e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 6.4210e+00
  7.8900e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9690e+02
  9.1400e+00]
 [2.7290e-02 0.0000e+00 7.0700e+00 0.0000e+00 4.6900e-01 7.1850e+00
  6.1100e+01 4.9671e+00 2.0000e+00 2.4200e+02 1.7800e+01 3.9283e+02
  4.0300e+00]
 [3.2370e-02 0.0000e+00 2.1800e+00 0.0000e+00 4.5800e-01 6.9980e+00
  4.5800e+01 6.0622e+00 3.0000e+00 2.2200e+02 1.8700e+01 3.9463e+02
  2.9400e+00]
 [6.9050e-02 0.0000e+00 2.1800e+00 0.0000e+00 4.5800e-01 7.1470e+00
  5.4200e+01 6.0622e+00 3.0000e+00 2.2200e+02 1.8700e+01 3.9690e+02
  5.3300e+00]]
[24.  21.6 34.7 33.4 36.2]
```

```
boston_data = pd.DataFrame(data=boston.data, columns=[x.lower() for x in boston.feature_names])
boston_data['medv'] = boston.target
boston_data.head()
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	medv
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	5.33	36.2

Рис. 1 (б). Обзор и подготовка данных в датасете «Цены на жилье в Бостоне».

Просмотр общей информации о данных в датафрейме:

```
boston_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    float64
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    float64
 9   tax      506 non-null    float64
 10  ptratio  506 non-null    float64
 11  b        506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(14)
memory usage: 55.5 KB
```

Рис. 2 (б). Общая информация о данных.

Заметим, что в столбцах *chas* и *rm* тип данных *float*, хотя по смыслу должен быть целочисленный *int*. Заменим типы, округлив при этом данные в столбце *rm* до целых.

```
boston_data['chas'] = boston_data['chas'].astype('int')
boston_data['rm'] = boston_data['rm'].round().astype('int')
print('Тип данных в столбце chas:', boston_data['chas'].dtype)
print('Тип данных в столбце rm:', boston_data['rm'].dtype)
```

```
Тип данных в столбце chas: int32
Тип данных в столбце rm: int32
```

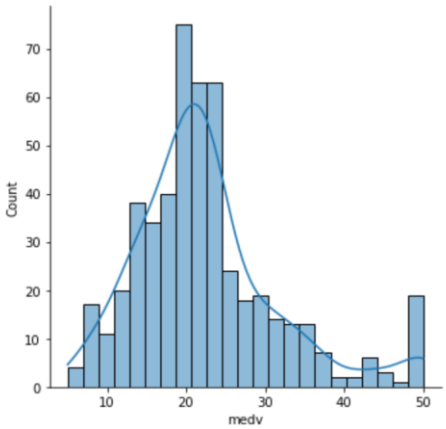
Рис. 3 (б). Замена типов данных.

Других проблем в датасете нет, данные готовы к дальнейшему исследованию.

Визуализация данных и нахождение зависимостей.

На данном этапе будем искать зависимость средней стоимости домов от остальных переменных. Для начала рассмотрим распределение целевых значений на графике и в числах.

```
sns.displot(boston_data['medv'], kde=True)
plt.show()
```




```
boston_data['medv'].describe()
```

```
count    506.000000
mean      22.532806
std        9.197104
min        5.000000
25%       17.025000
50%       21.200000
75%       25.000000
max       50.000000
Name: medv, dtype: float64
```

Рис. 4 (б). Распределение значений целевой переменной.

Распределение похоже на нормальное, однако максимум значений немного смещен в меньшую сторону, а еще видим ощутимый скачок

количества домов стоимостью 50 тысяч долларов – это больше похоже на выброс, который может помешать в исследовании. Также заметим, что три четверти домов находятся в ценовом сегменте до 25 тыс. долл.

Удалим из таблицы все дома стоимостью 50 тыс. долл. и построим тепловую карту корреляции столбцов.

```
boston_data = boston_data[boston_data['medv'] != 50.0].reset_index(drop=True)
```

```
corr = boston_data.corr().round(2)
mask = np.triu(np.ones_like(corr, dtype=np.bool))
plt.figure(figsize=(10, 10))
sns.heatmap(corr, mask=mask, vmin=-1, vmax=1, annot=True, cmap='BrBG').set_title(
                            'Тепловая карта корреляции', fontdict={'fontsize':12});
```

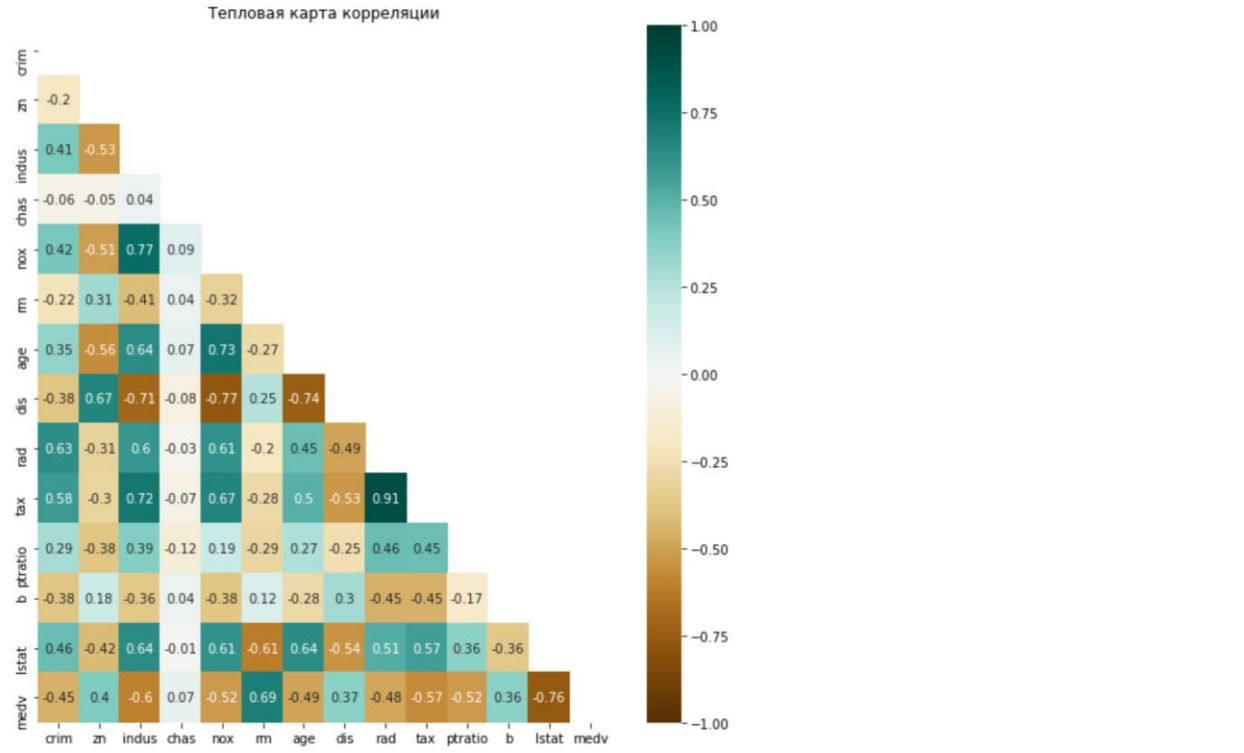


Рис. 5 (б). Тепловая карта корреляции.

Исходя из этой таблицы, можно сделать следующие выводы:

- параметры *rad* и *tax* имеют наибольшую взаимосвязь – их коэффициент корреляции равен 0.91;
- наибольшую связь с целевой переменной имеют характеристики *rm* (довольно высокая положительная корреляция) и *lstat* (большая отрицательная корреляция).

Выберем параметры *rm* и *lstat* и будем использовать их для обучения модели. Сначала продемонстрируем связь этих характеристик и средней цены домов.

```
plt.figure(figsize=(20, 5))

features = ['rm', 'lstat']
target = boston_data['medv']

for i, col in enumerate(features):
    plt.subplot(1, len(features), i + 1)
    x = boston_data[col]
    y = target
    plt.scatter(x, y, marker='o')
    plt.xlabel(col)
    plt.ylabel('medv')
```

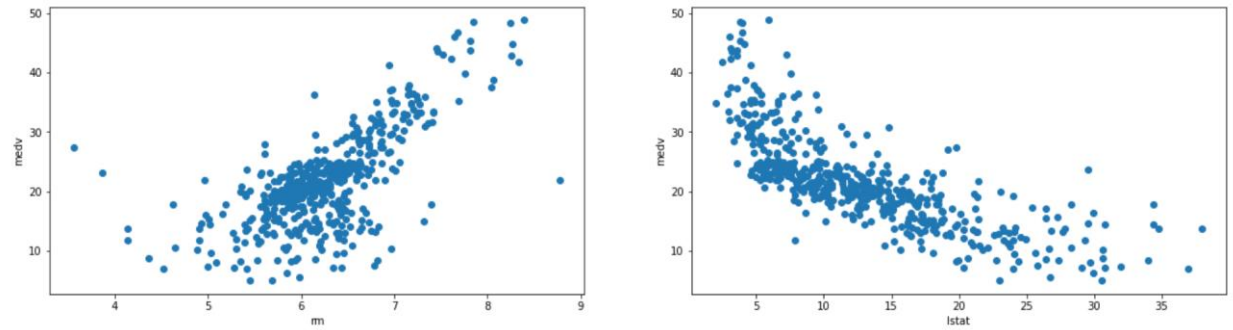


Рис. 6 (б). Точечные графики зависимости цены домов.

Первый график иллюстрирует положительную корреляцию *rm* и *medv*. Вполне ожидаемо - с ростом количества комнат в доме увеличивается и цена; и хотя присутствуют значения, выпадающие из общей массы, основная часть рассредоточена вдоль одной линии.

Второй график демонстрирует отрицательную корреляцию параметров *lstat* и *medv*. Тоже предсказуемо - в районах, где мало людей с низким социальным статусом, недвижимость довольно дорогая, а где их больше, там и жилье подешевле. Несмотря на это, график лишь с натяжкой похож на линейный, скорее на гиперболу.

Построение и обучение моделей.

Для данной задачи изучим 3 различных модели регрессионного анализа: обычную линейную регрессию *LinearRegression*, линейную модель со стохастическим градиентным спуском *SGDRegressor* и алгоритм "случайного леса" *RandomForestRegressor*.

Обучим эти модели, чтобы предсказать целевые данные. Разобьем набор данных на обучающее и проверочное множества в отношении 4:1, установив параметр *test_size=0.2*.

Для оценки качества моделей используем регрессионные метрики (иначе, функции потерь) - корень из средней квадратической ошибки *mean_squared_error* и коэффициент детерминации *r2_score*.

Средняя квадратическая ошибка рассчитывается как сумма квадратов разностей между реальным и предсказанным значением, деленная на число значений. Коэффициент детерминации получается путем вычитания из единицы отношения суммы квадратов остатков регрессии к сумме квадратов разностей между фактическим значением и средним.

```
data_train, data_test, medv_train, medv_test = train_test_split(
    boston_data[['rm', 'lstat']], boston_data['medv'], test_size=0.2, random_state=42)

for regressor in [linear_model.LinearRegression(),
                  linear_model.SGDRegressor(),
                  ensemble.RandomForestRegressor()]:
    model = regressor
    model.fit(data_train, medv_train)
    train_predict = model.predict(data_train)
    rmse = np.sqrt(metrics.mean_squared_error(medv_train, train_predict))
    r2 = metrics.r2_score(medv_train, train_predict)
    print(f"Метрики качества модели {regressor} (тренировочное множество):")
    print(f'Корень из среднеквадратической ошибки = {rmse:.3f}')
    print(f'Коэффициент детерминации = {r2:.3f}')
    print()
    test_predict = model.predict(data_test)
    rmse = np.sqrt(metrics.mean_squared_error(medv_test, test_predict))
    r2 = metrics.r2_score(medv_test, test_predict)
    print(f"Метрики качества модели {regressor} (тестовое множество):")
    print(f'Корень из среднеквадратической ошибки = {rmse:.3f}')
    print(f'Коэффициент детерминации = {r2:.3f}')
    print()
```

```
Метрики качества модели LinearRegression() (тренировочное множество):
Корень из среднеквадратической ошибки = 4.651
Коэффициент детерминации = 0.660

Метрики качества модели LinearRegression() (тестовое множество):
Корень из среднеквадратической ошибки = 4.438
Коэффициент детерминации = 0.614

Метрики качества модели SGDRegressor() (тренировочное множество):
Корень из среднеквадратической ошибки = 5.845
Коэффициент детерминации = 0.463

Метрики качества модели SGDRegressor() (тестовое множество):
Корень из среднеквадратической ошибки = 5.140
Коэффициент детерминации = 0.483

Метрики качества модели RandomForestRegressor() (тренировочное множество):
Корень из среднеквадратической ошибки = 1.452
Коэффициент детерминации = 0.967

Метрики качества модели RandomForestRegressor() (тестовое множество):
Корень из среднеквадратической ошибки = 3.378
Коэффициент детерминации = 0.777
```

Рис. 7 (б). Обучение и тестирование моделей.

По итогам сравнения моделей оказалось, что для выбранных параметров наиболее точно предсказывает цену домов регрессор на основе алгоритма «случайного леса»: на тренировочном множестве коэффициент детерминации близок к 1, а на тестовом – 0.777, но это все равно ощутимо лучше, чем линейные модели. Вероятно, для данной задачи линейные модели использовать нецелесообразно, поскольку параметры, которые больше всего коррелируют с целевой переменной, имеют абсолютное значение коэффициента корреляции порядка 0.6–0.7, что не так уж и близко к полной линейной зависимости.

5. Заключение.

В данной курсовой работе мной изучены методы интеллектуального анализа, такие как классификация и регрессионный анализ.

Проведен интеллектуальный анализ датасета «Ирисы Фишера», построена и обучена модель-классификатор, которая может предсказывать вид ириса по заданным параметрам.

Проведен интеллектуальный анализ датасета «Цены на жилье в Бостоне», построены и обучены 3 различных регрессионных модели, которые могут с определенной точностью предсказывать цену объекта жилья по заданным параметрам, произведено сравнение моделей.

В каждом случае была получена хорошая точность прогноза. В дальнейшем можно получить еще более высокую точность, обучая различные модели и применяя еще более разнообразные методы повышения качества обучения, чтобы приблизить точность прогноза и другие метрики качества к 100%, однако главной целью данной курсовой работы было изучить и применить на практике основные методы ИАД, которая, на мой взгляд, была достигнута.

6. Список источников.

1. Маккини У. Python и анализ данных / пер. с анг. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 540 с.: ил. – ISBN 978-5-94074-590-5.

2. Степанов Р. Г. Технология Data Mining: Интеллектуальный Анализ Данных – Казань: [б. и.], 2008. – 58 с.

3. Дюк В. Data Mining: учебный курс (+CD) / Дюк В., Самойленко А. — СПб.: Питер, 2001. — 368 с.: ил. – ISBN 5-318-00227-7.

4. Pandas documentation – pandas 1.3.5 documentation. – 2008. – URL: https://pandas.pydata.org/docs/ (дата обращения: 29.11.2021).

5. NumPy documentation – NumPy v1.22 Manual. – 2008. – URL: https://numpy.org/doc/stable/ (дата обращения: 29.11.2021).

6. Matplotlib documentation — Matplotlib 3.5.1 documentation. – 2002. – URL: https://matplotlib.org/stable/index.html (дата обращения: 29.11.2021).

7. Seaborn: statistical data visualization – seaborn 0.11.2 documentation. – 2012. – URL: https://seaborn.pydata.org/ (дата обращения: 29.11.2021).

8. User guide: contents – scikit-learn 1.0.2 documentation. – 2007. – URL: https://scikit-learn.org/stable/user_guide.html (дата обращения: 29.11.2021).

9. НОУ ИНТУИТ | Data Mining. – Москва, 2003. – URL: https://intuit.ru/studies/courses/6/6/info (дата обращения: 05.01.2022).

10. T. Shin. How I Consistently Improve My Machine Learning Models From 80% to Over 90% Accuracy // Towards Data Science. – 2020. – URL: https://towardsdatascience.com/how-i-consistently-improve-my-machine-learning-models-from-80-to-over-90-accuracy-6097063e1c9a (дата обращения: 06.01.2022).