Министерство науки и высшего образования РФ

Томский государственный университет
систем управления и радиоэлектроники

Е.Е. Черявко, К.С. Саксонов, С.А. Литовкин, Е.Ю. Костюченко

НЕЙРОННЫЕ СЕТИ В БИОМЕДИЦИНСКИХ ТЕХНОЛОГИЯХ

Методические указания к лабораторным работам
для студентов направлений подготовки
09.04.04 Программная инженерия

УДК 004.8
ББК 32.813.5
Л 64

Литовкин. С.А. НЕЙРОННЫЕ СЕТИ В БИОМЕДИЦИНСКИХ ТЕХНОЛОГИЯХ: Учебно-методическое пособие / Е. Ч. Черявко, К.С. Саксонов, С.А. Литовкин, Е.Ю. Костюченко. — Томск: ТУСУР, 2024. — 87 с.

Настоящие методические указания содержит описания лабораторных и самостоятельных работ по дисциплине «Нейронные сети в биомедицинских технологиях» для направлений подготовки, входящих в укрупненную группу специальностей и направлений 09.04.04 Программная инженерия.

Одобрено на заседании кафедры КИБЭВС протокол №7 от 30.08.2024 года

УДК 004.8
ББК 32.813.5

Содержание

Введение 4
ЛАБОРАТОРНАЯ РАБОТА №1 5
ЛАБОРАТОРНАЯ РАБОТА № 2 23
ЛАБОРАТОРНАЯ РАБОТА №3 34
ЛАБОРАТОРНАЯ РАБОТА № 4 46
ЛАБОРАТОРНАЯ РАБОТА №5 54
ЛАБОРАТОРНАЯ РАБОТА № 6 64
ЛАБОРАТОРНАЯ РАБОТА № 7 71
Литература 81

Введение

Целью преподавания дисциплины является изучение принципов работы нейронных сетей и применения их на практике.

Задачи изучения дисциплины:

– Изучить основные принципы машинного обучения и нейронных сетей.

– Рассмотреть принципы применения машинного обучения и нейронных сетей для задач анализа данных.

– Рассмотреть принципы применения языка программирования Python и существующие библиотеки для машинного обучения и нейронных сетей.

–

ЛАБОРАТОРНАЯ РАБОТА №1

Практическое применение линейной и логистической регрессий

Целью работы является изучение и практическое применение методов линейной и логистической регрессий для анализа и прогнозирования данных.

Краткие теоретические сведения

Регрессия - это статистический подход, используемый для анализа взаимосвязи между зависимой переменной (целевой переменной) и одной или несколькими независимыми переменными (предсказуемыми переменными).

Цель состоит в том, чтобы определить наиболее подходящую функцию, которая характеризует связь между этими переменными. Она направлена на поиск наиболее подходящей модели, которую можно использовать для прогнозирования или выводов.

Линейная регрессия - это тип контролируемого алгоритма машинного обучения, который вычисляет линейную зависимость между зависимой переменной и одним или несколькими независимыми признаками путем подгонки линейного уравнения к наблюдаемым данным. Когда имеется только один независимый признак, это называется простой линейной регрессией, а когда имеется более одного признака, это называется множественной линейной регрессией.

Аналогично, когда имеется только одна зависимая переменная, это считается одномерной линейной регрессией, в то время как когда имеется более одной зависимой переменной, это известно, как многомерная регрессия.

Линейная регрессия — это не просто инструмент прогнозирования; она формирует основу для различных продвинутых моделей. Такие методы, как регуляризация и машины опорных векторов, черпают вдохновение из линейной

регрессии, расширяя ее полезность. Кроме того, линейная регрессия является краеугольным камнем в тестировании допущений, позволяя исследователям проверять ключевые предположения о данных.

Основная цель при использовании линейной регрессии - найти наиболее подходящую линию, что подразумевает, что ошибка между прогнозируемыми и фактическими значениями должна быть сведена к минимуму. В наиболее подходящей линии будет наименьшая ошибка.

Линейное уравнение наилучшего соответствия представляет собой прямую линию, представляющую взаимосвязь между зависимой и независимой переменными. Наклон линии показывает, насколько сильно изменяется зависимая переменная при единичном изменении независимой переменной (переменных).

Матрица запутанности (Confusion Matrix) — это метрика, которая показывает количество правильных и ошибочных предсказаний для каждого класса.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также представленными примерами в Google Colab.
2. Выполнить ход работы.
3. Выполнить индивидуальное задание.
4. Оценить полученные результаты.
5. Ответить на контрольные вопросы.

1 Ход работы

1.1 Загрузка данных

Перед началом работы необходимо скачать набор данных: https://www.kaggle.com/datasets/spittman1248/cdc-data-nutrition-physical-activity-obesity?resource=download. Он включает данные о рационе питания, физической активности и весе взрослых из системы наблюдения за поведенческими факторами риска. После того как набор данных будет скачан, его необходимо загрузить в файловое пространство Colab.

Для загрузки файла с локального компьютера в Google Colab можно воспользоваться следующим кодом:

```
from google.colab import files
uploaded = files.upload()

data = pd.read_csv('Nutrition__Physical_Activity__and_Obesity_-
_Behavioral_Risk_Factor_Surveillance_System.csv')

# Просмотр первых строк данных

data.head()

# Просмотр первых строк данных и информации о них
print("Первые строки данных:")
print(data.head())
print("\nИнформация о данных:")
print(data.info())
```

1.2 Предварительный анализ данных

Для предварительного анализа данных выполните следующие действия: сначала определите количество пропущенных значений в столбцах Data_Value, Education, Income и Gender. Рассчитайте процент этих значений относительно общего числа строк. Затем удалите строки, содержащие пропущенные значения в указанных столбцах, используя метод dropna с параметром subset. После этого проверьте, остались ли пропущенные значения в других столбцах. Выведите

уникальные значения для столбцов Education и Income, а также определите их типы данных.

```
# Фильтрация данных: сохранять только строки, в которых значения
«Data_Value», Education и Income не равны нулю.
data_complete = data.dropna(subset=['Data_Value', 'Education', 'Income',
'Gender'])

# Проверка наличия пропущенных значений
print("\nПроверка на пропущенные значения:")
print(data.isnull().sum())

# Проверка уникальных значений в разделах 'Education' и 'Income'
unique_education = data_complete['Education'].unique()
unique_income = data_complete['Income'].unique()

# Проверка типов данных 'Education' и 'Income'
dtype_education = data_complete['Education'].dtype
dtype_income = data_complete['Income'].dtype

(unique_education, dtype_education, unique_income, dtype_income)
```

Выполните подсчет ненулевых значений и уникальных значений для каждого столбца, затем создайте DataFrame, который будет хранить информацию, подсчитанную выше. Получите список переменных для прогнозирования через фильтрацию DataFrame, оставив только переменные содержащие более 70% ненулевых значений и не менее 2 уникальных значений.

```
# Подсчет ненулевых значений и уникальных значений для каждого столбца
non_null_counts = data.notnull().sum()
unique_counts = data.nunique()

# Создание DataFrame для отображения информации
variable_info = pd.DataFrame({
    'non_null_counts': non_null_counts,
    'unique_counts': unique_counts
})

# Фильтровать переменные с более чем 70% ненулевыми значениями и не менее
чем с 2 уникальными значениями (чтобы избежать констант)
potential_predictors = variable_info[(variable_info['non_null_counts'] > 0.7
* len(data)) & (variable_info['unique_counts'] > 1)]

# Показать потенциальные переменные для прогнозирования
potential_predictors.sort_values(by='non_null_counts', ascending=False)
```

Визуализируйте данные (рисунок 1.1 – 1.4).

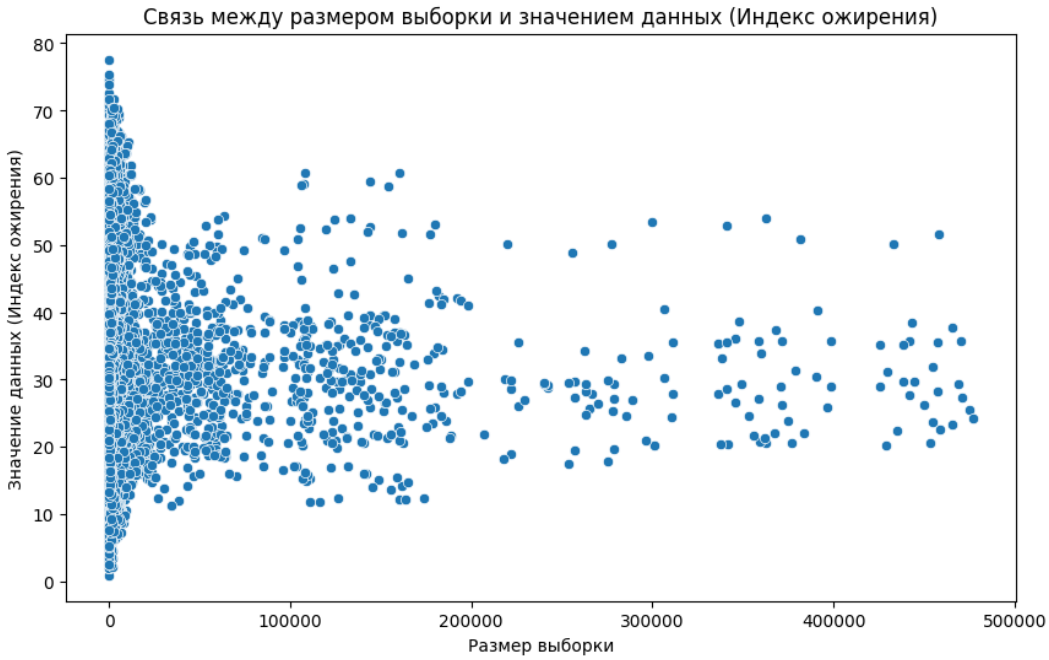

Рисунок 1.1 – Визуализация связи между «Sample_Size» и «Data_Value»

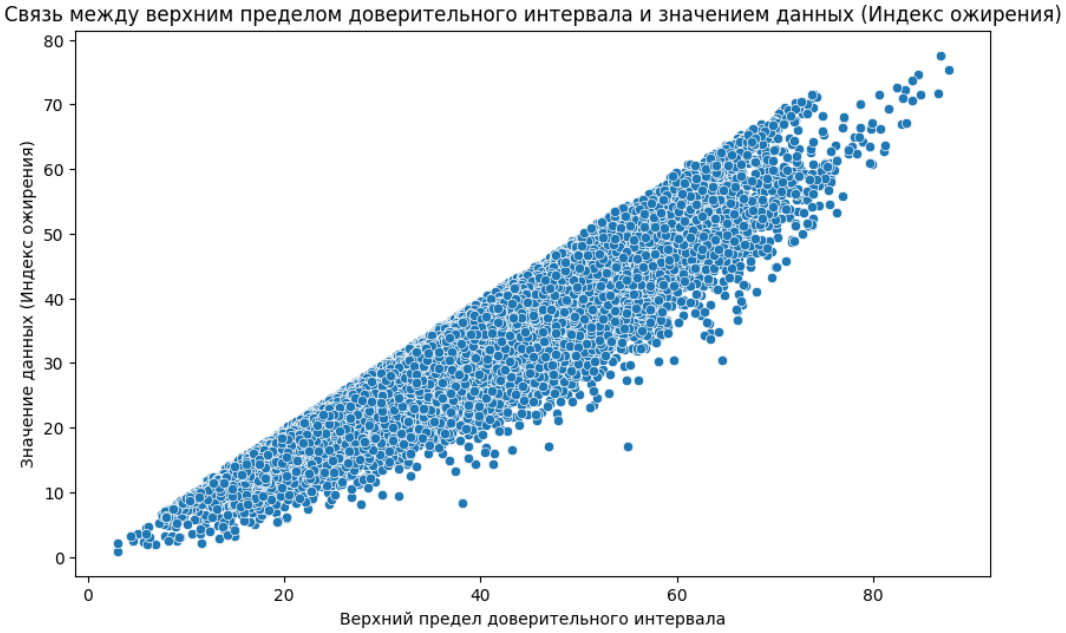

Рисунок 1.2 – Визуализация связи между «High_Confidence_Limit» и «Data_Value»

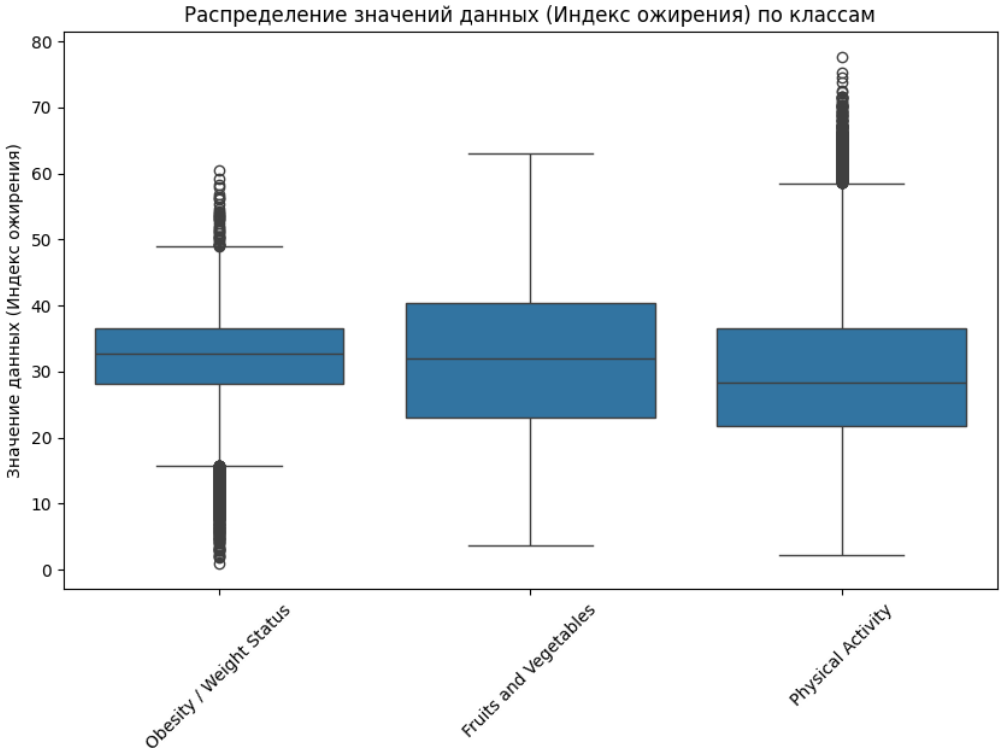

Рисунок 1.3 – Просмотр распределения «Data_Value» по «Class»

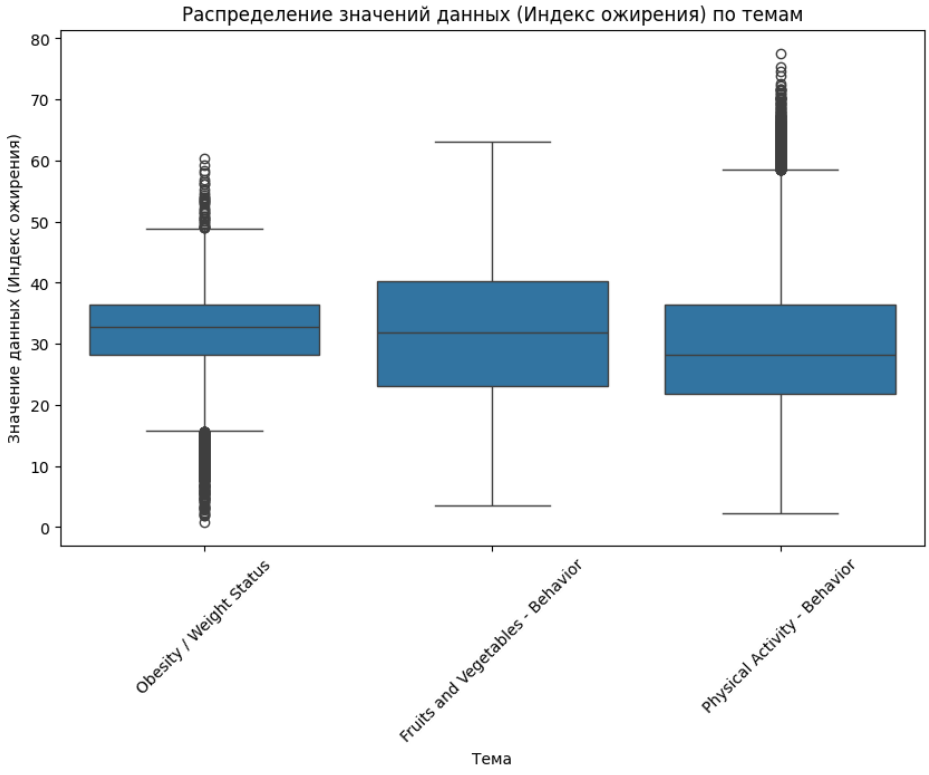

Рисунок 1.4 – Просмотр распределения «Data_Value» по «Topic»

1.3 Работа с моделью

Выберите числовые и категориальные переменные, после чего удалите строки с нулевыми значениями в «Data_Value». Затем разделите данные на обучающую и тестовую выборки.

```
# Выбранные переменные
num_vars = ['Low_Confidence_Limit', 'Sample_Size']
cat_vars = ['Class', 'Topic']

# Удалить строки с нулевыми значениями в 'Data_Value'
data_model = data.dropna(subset=['Data_Value'])

# Разделение данных на обучающие и тестовые наборы
X = data_model[num_vars + cat_vars]
y = data_model['Data_Value']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=0)
```

Далее создайте преобразователи для данных.

Для подготовки данных выполните следующие шаги:

1. Заполните пропущенные значения:
   a. Для числовых переменных — средним значением.
   б. Для категориальных переменных — наиболее частым значением.
2. Нормализуйте числовые переменные с помощью StandardScaler.
3. Закодируйте категориальные переменные с использованием метода OneHotEncoder.
4. Создайте преобразователи
5. Объедините преобразователи в единый предобработчик.

```
# Создание преобразователей для числовых и категориальных переменных
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean')),  # Заполнение пропущенных
значений средним
    ('scaler', StandardScaler())  # Нормализация данных
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Заполнение
пропущенных значений наиболее частым
    ('onehot', OneHotEncoder(handle_unknown='ignore'))  # Преобразование
категориальных переменных в бинарные
])
# Комбинирование преобразователей в единый предобработчик
preprocessor = ColumnTransformer(
    transformers=[
```

```
        ('num', numeric_transformer, num_vars),
        ('cat', categorical_transformer, cat_vars)
    ])
```

Далее создайте модель линейной регрессии и обучите её на обучающих данных. После завершения обучения выполните прогнозирование значений на тестовой выборке.

```
# Определение модели
model = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('regressor', LinearRegression())
])

# Обучение модели
model.fit(X_train, y_train)
# Прогнозирование значений на тестовых данных
y_pred = model.predict(X_test)

# Оценка модели
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

(mae, mse, r2)
```

Также визуализируйте значения предсказаний и реальных значений для сравнения (рисунок 1.5).

Рисунок 1.5 – Визуализация реальные значения vs предсказания

Далее построим и обучим модель с использованием TensorFlow/Keras. Данные преобразуются с использованием ранее определенного преобразователя. После чего создается последовательная модель, а также добавляется полносвязный слой с одним нейроном и заданным размером входного слоя.

Далее модель компилируется с использованием оптимизатора Adam и функции потерь mean_squared_error. Затем происходит обучение модели. При этом указываются: количество эпох (epochs), доля данных, выделяемая для валидационного набора (validation_split), и уровень детализации вывода (verbose). Параметр verbose=0 отключает вывод подробностей о процессе обучения на каждой эпохе.

```
# Построение модели с использованием TensorFlow/Keras
model_tf = models.Sequential([
    layers.Dense(1, input_dim=X_train_preprocessed.shape[1])
])

# Компиляция модели
model_tf.compile(optimizer='adam', loss='mean_squared_error')

# Обучение модели
history = model_tf.fit(
    X_train_preprocessed, y_train,
    epochs=50,
    validation_split=0.2,  # Использовать часть обучающего набора для
валидации
    verbose=0  # Не выводить подробности каждой эпохи
)
```

Выполните предсказание значений на тестовых данных и преобразование результата в одномерный массив. После чего рассчитайте метрики для оценки качества модели TensorFlow и получите значений функции потерь для последней эпохи на обучающей и валидационной выборках.

После обучения визуализируйте значения предсказаний и реальных значений для сравнения, но уже с использованием TensorFlow (рисунок 1.6).

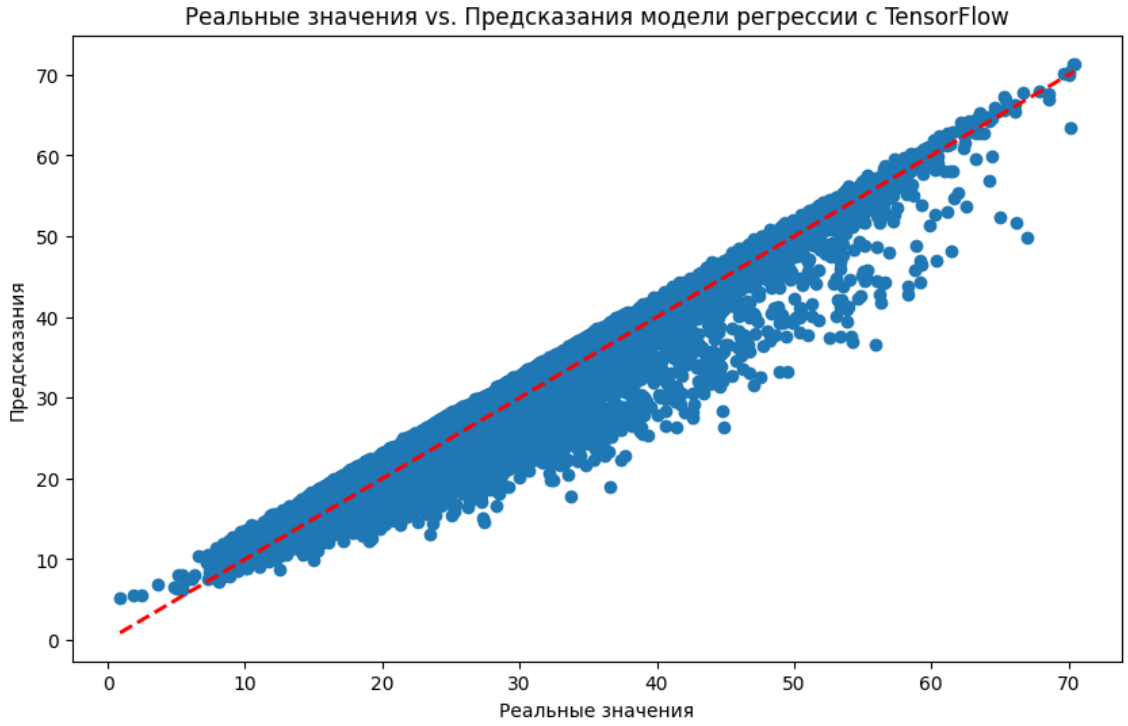

Рисунок 1.6 – Визуализация предсказаний с TensorFlow vs реальные значения

1.4 Логистическая регрессия

Для работы с логистической регрессией выполните аналогичные этапы, как и для линейной регрессии. Проведите предварительную обработку данных, выберите переменные, которые будут использоваться в модели, и разделите данные на обучающие и тестовые наборы.

Создайте преобразователи для числовых и категориальных переменных, скомбинируйте их в единый предобработчик и определите модель — в данном случае это логистическая регрессия. Обучите модель и произведите прогнозирование значений на тестовых данных. Затем оцените модель и выведите результаты.

В завершение визуализируйте матрицу запутанности и постройте ROC-кривую (рисунки 1.7 – 1.8).

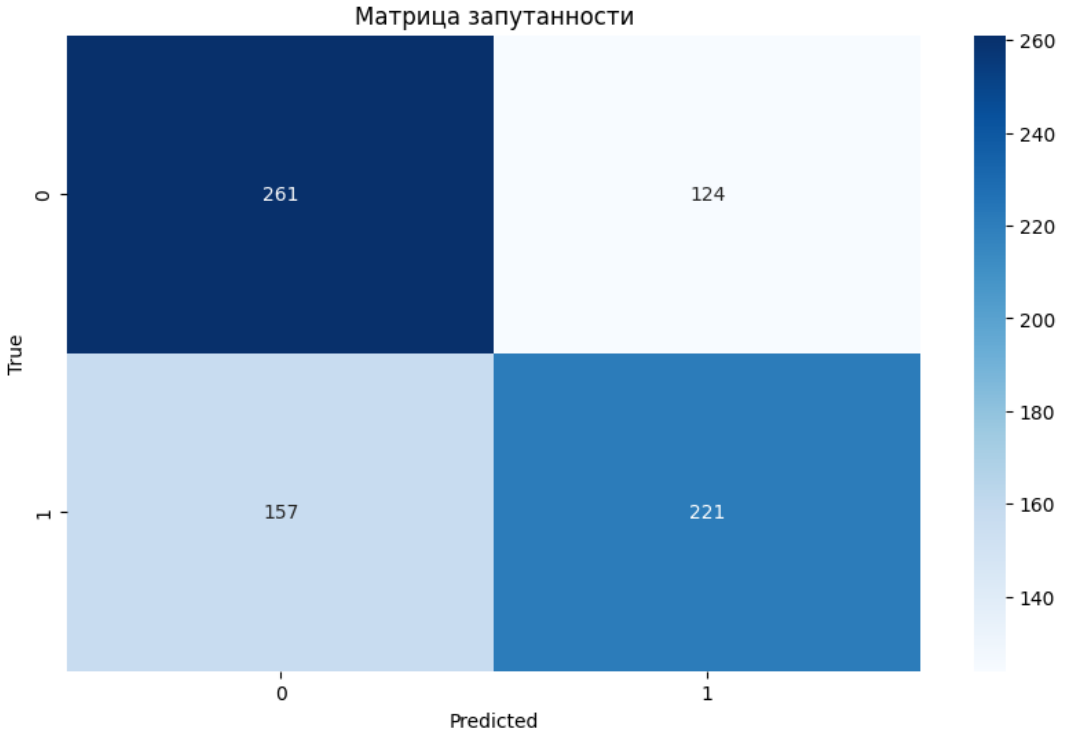

Рисунок 1.7 – Визуализация матрицы запутанности

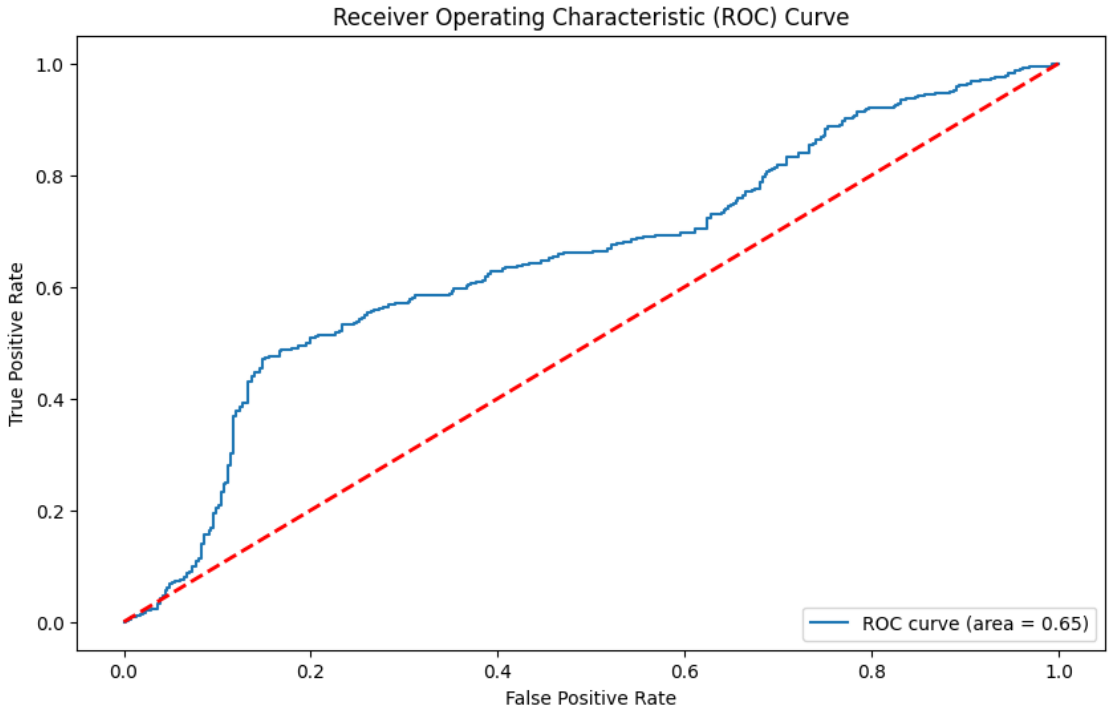

Рисунок 1.8 – Визуализация ROC-кривой

1.5 Логистическая регрессия на других данных

Проведите работу с логистической регрессией в другом блокноте и на другом наборе данных. Перед началом работы с новым набором данных необходимо скачать набор данных: https://www.kaggle.com/datasets/dragonheir/logistic-regression?resource=download.

После того как набор данных будет скачан, его необходимо загрузить в файловое пространство Colab. Загрузите данные из CSV файла в DataFrame и проведите первичный анализ.

```
df = pd.read_csv('/kaggle/input/logistic-regression/Social_Network_Ads.csv')
df.head()
df.info()
df.describe()
df.isnull().all()
```

Выведите графики, чтобы визуально исследовать распределение возраста и предполагаемой зарплаты (рисунок 1.9).

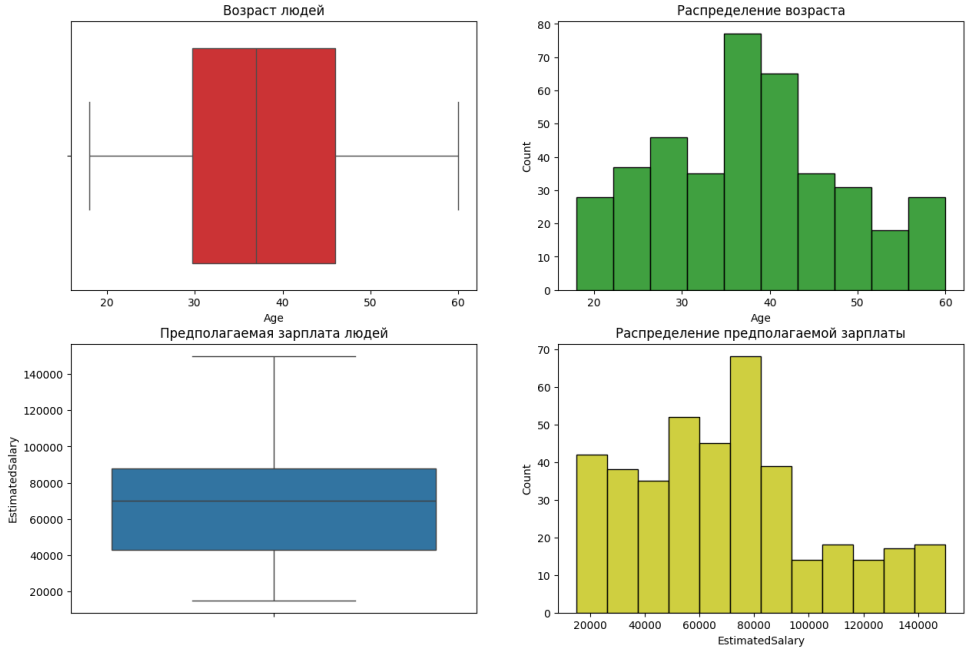

Рисунок 1.9 – Распределение возраста и предполагаемой зарплаты

Выведите графики, которые показывают распределение зарплаты и возраста по полу (рисунок 1.10).

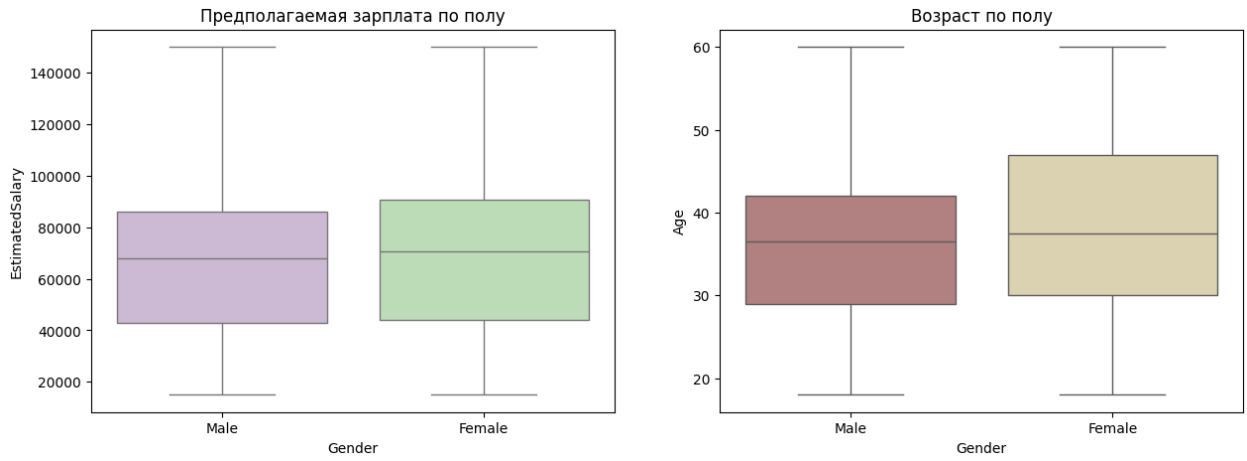

Рисунок 1.10 – Распределение зарплаты и возраста по полу

Выведите графики, которые показывают количество покупок и их распределение по полу (рисунок 1.11).

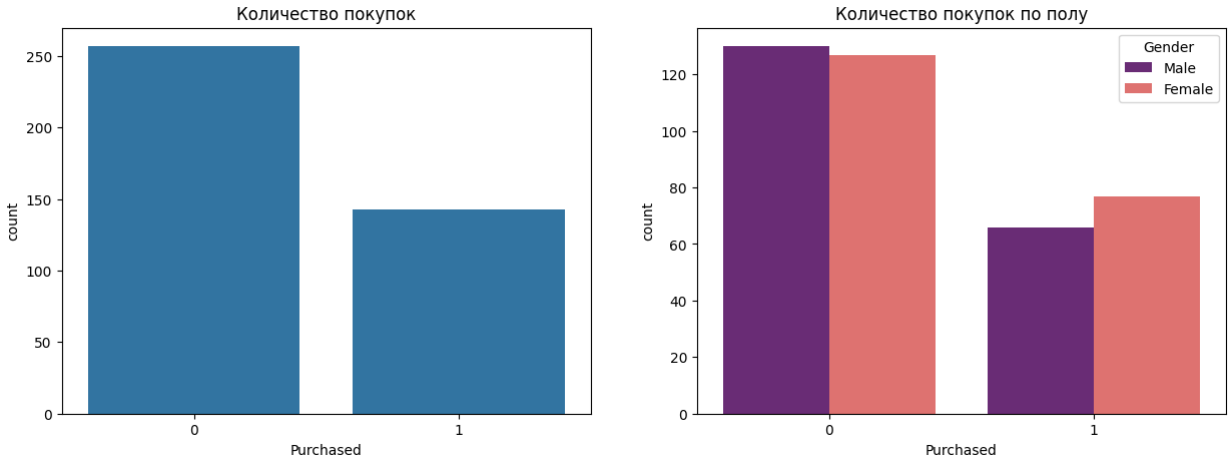

Рисунок 1.11 – Количество покупок и их распределение по полу

Определите корреляцию данных, после чего создайте тепловую карту для визуализации корреляций между переменными (рисунок 1.12).

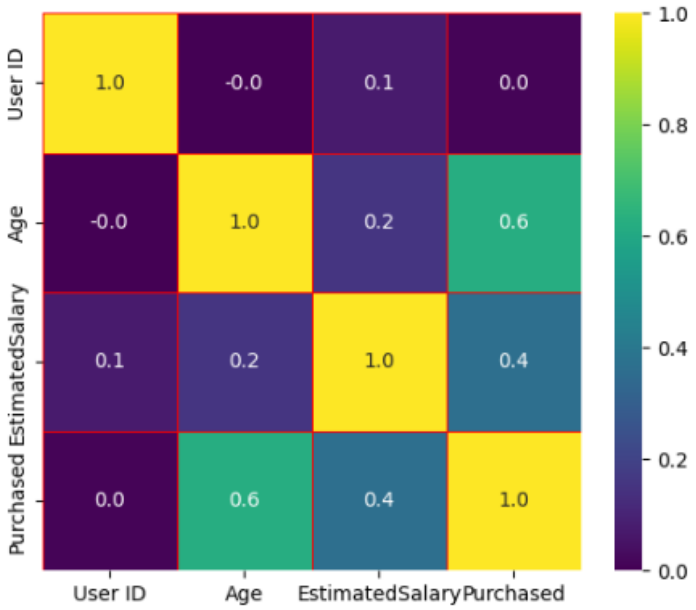

Рисунок 1.12 – Тепловая карта

Удалите столбец User ID, так как он не несет полезной информации для обучения модели. Преобразуйте значения в столбце Gender в числовой формат (например, замените «Male» на 1 и «Female» на 0), чтобы упростить их обработку. Проведите нормализацию данных для приведения всех признаков к единой шкале, что улучшит работу модели.

Для разделения данных на обучающую и тестовую выборки используйте метод train_test_split из библиотеки sklearn. Необходимо разделить данные таким образом, чтобы 80% данных использовалось для обучения модели, а 20 % – для тестирования.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(d_scaled, y,
test_size=0.20, random_state=42)
```

Для построения модели используйте алгоритм логистической регрессии. Задайте параметр регуляризации *C*=0.1 и максимальное количество итераций – 500, после чего обучите модель.

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression(C=0.1, max_iter=500)
model.fit(X_train, y_train)

y_pred = model.predict(X_test)
```

После выполнения расчетов необходимо вывести следующие метрики:

– Accuracy
– Precision
– Recall
– F1 Score

Затем сравните полученные значения метрик и обратите внимание на соотношение Recall и Precision. Если Recall окажется ниже Precision, это может указывать на возможные пропуски положительных случаев.

Далее создайте и выведите матрицу запутанности (confusion matrix) для более детального анализа результатов, матрица запутанности представлена на рисунке 1.13.

```
from sklearn.metrics import confusion_matrix

df = pd.DataFrame(confusion_matrix(y_test, y_pred), columns=['Predicted
Positive', 'Predicted Negative'],
                  index=['Actual Positive', 'Actual Negative'])
df
```

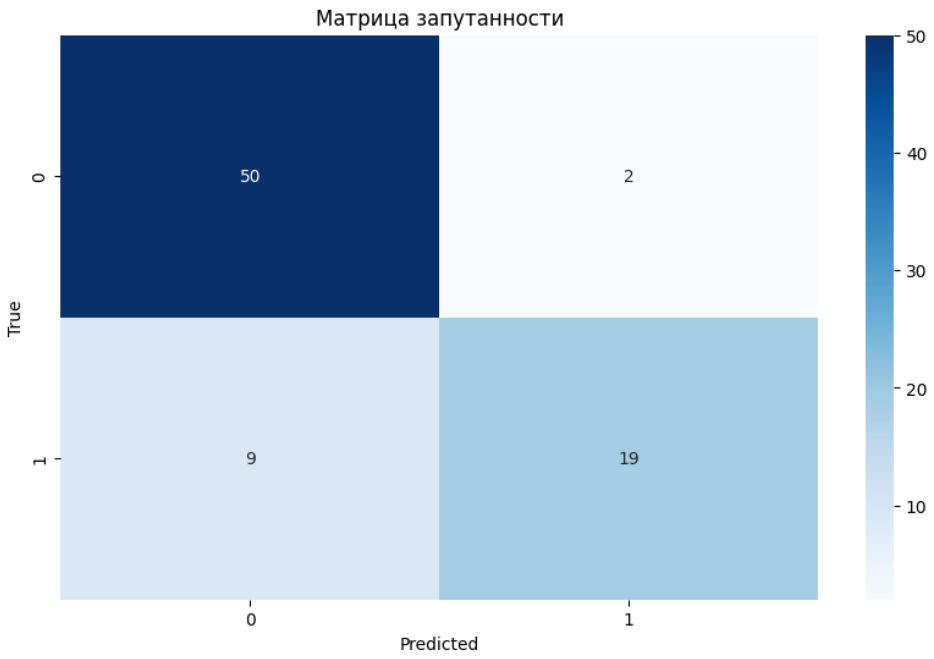

Рисунок 1.13 – Матрица запутанности

Постройте ROC-кривую и вычислите площадь под кривой (AUC) (рисунок 1.14).

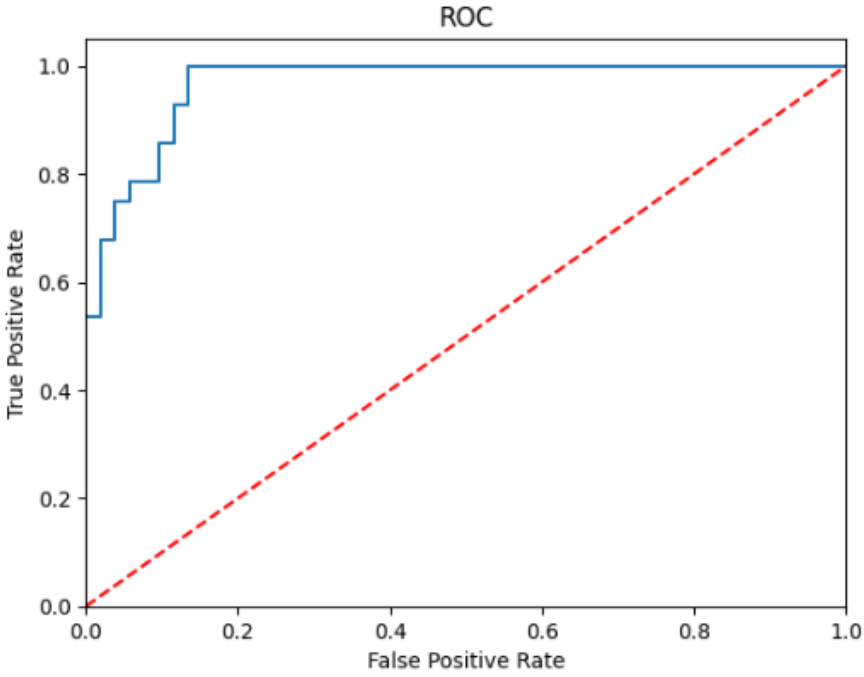

Рисунок 1.14 – ROC-кривая

После чего можно применить метод SMOTE для увеличения количества примеров миноритарного класса, то есть класса, представленного меньшим количеством объектов в наборе данных по сравнению с другими классами. Затем выполните повторное обучение модели на сбалансированных данных.

```
from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

clf = LogisticRegression()
model_res = clf.fit(X_train_res, y_train_res)
print(f'Test accuracy {model_res.score(X_test, y_test)}')

 from imblearn.over_sampling import SMOTE

sm = SMOTE(random_state=2)
X_train_res, y_train_res = sm.fit_resample(X_train, y_train.ravel())

clf = LogisticRegression()
model_res = clf.fit(X_train_res, y_train_res)
print(f'Test accuracy {model_res.score(X_test, y_test)}')

print(f'Originally: {X_train.shape}')
print(f'With SMOTE: {X_train_res.shape}')
```

После применения метода SMOTE необходимо рассчитать значение accuracy. При успешном выполнении операции ожидается получение значения, близкого к 0.9.

2 Индивидуальное задание

После выполнения основной работы необходимо выполнить задания, представленные в таблице 2.1.

В них требуется модифицировать нейронную сеть в соответствии с указанными параметрами и сравнить результаты с исходной моделью.

Также следует изменить параметры, предложенные для вашего задания, и провести сравнительный анализ полученных результатов.

Таблица 2.1 – Задания по вариантам

Вариант	Архитектура нейронной сети и параметры	1 набор	2 набор
1	2 Dense слоя (128 узлов) с 1 Dropout(0.2), активация: ReLU, оптимизатор: Adam	Линейная регрессия, с параметрами разбиения данных: test_size=0.1 и 0.8	Логистическая регрессия, C=1.0, max_iter=100, test_size=0.3 и 0.65
2	3 Dense слоя (64 узла) с 2 Dropout(0.3), активация: Sigmoid, оптимизатор: SGD	Логистическая регрессия, с параметрами разбиения данных: C=0.5, max_iter=200, test_size=0.15 и 0.85	Логистическая регрессия, C=0.5, max_iter=200, test_size=0.2 и 0.7
3	1 Dense слой (256 узлов) с 1 Dropout(0.1), активация: Tanh, оптимизатор: RMSprop	Линейная регрессия, с параметрами разбиения данных: test_size=0.2 и 0.75	Логистическая регрессия, C=1.5, max_iter=150, test_size=0.25 и 0.6
4	4 Dense слоя (32 узла) с 3 Dropout(0.4), активация: ReLU, оптимизатор: Adam	Логистическая регрессия, с параметрами: C=0.1, max_iter=300, test_size=0.1 и 0.9	Логистическая регрессия, C=2.0, max_iter=250, test_size=0.15 и 0.5
5	2 Dense слоя (128 узлов) с 2 Dropout(0.25), активация: ReLU, оптимизатор: Adam, Batch Normalization	Линейная регрессия, с параметрами разбиения данных: test_size=0.2 и 0.8	Логистическая регрессия, C=0.75, max_iter=180, test_size=0.3 и 0.55
6	3 Dense слоя (64 узла) с 1 Dropout(0.15), активация: Sigmoid, оптимизатор: Adagrad	Логистическая регрессия, с параметрами: C=0.75, max_iter=220, test_size=0.05 и 0.95	Логистическая регрессия, C=1.25, max_iter=130, test_size=0.2 и 0.65
7	1 Dense слой (512 узлов) с 1 Dropout(0.05), активация:	Линейная регрессия, с параметрами разбиения	Логистическая регрессия, C=1.75,

	ReLU, оптимизатор: Adam	данных: test_size=0.25 и 0.7	max_iter=170, test_size=0.25 и 0.7
8	5 Dense слоев (16 узлов) с 4 Dropout(0.35), активация: Tanh, оптимизатор: Nadam	Логистическая регрессия, с параметрами: C=0.3, max_iter=250, test_size=0.1 и 0.85	Логистическая регрессия, C=0.25, max_iter=300, test_size=0.1 и 0.6

Контрольные вопросы

1. Что такое линейная регрессия?
2. Что такое логистическая регрессия?
3. Какова основная цель предобработки данных?
4. В чем разница между регрессией и классификацией?
5. Какие метрики используются для оценки качества регрессионных моделей?
6. Какие метрики используются для оценки качества классификационных моделей?
7. Что такое матрица запутанности?

ЛАБОРАТОРНАЯ РАБОТА № 2

Практическое применение линейного дискриминантного анализа, k-ближайших соседей и наивного байесовского классификатора.

Целью лабораторной работы является изучение и практическое применение линейного дискриминантного анализа (LDA), метода k-ближайших соседей (k-NN) и наивного байесовского классификатора (Naive Bayes) для решения задачи классификации.

Краткие теоретические сведения

Линейный дискриминантный анализ (**LDA**) — это метод классификации и снижения размерности, который позволяет эффективно разделять классы.

Основная идея LDA заключается в предположении о многомерном нормальном распределении признаков внутри классов и поиске такого линейного преобразования, которое максимизирует межклассовую дисперсию и минимизирует внутриклассовую.

Иными словами, объекты разных классов должны иметь нормальное распределение и быть максимально удалены друг от друга, а объекты одного класса — сгруппированы вместе.

Принцип работы **LDA**:

1. Изначально для всех классов рассчитываются априорные вероятности и средние значения признаков.
2. На основе полученных значений рассчитываются ковариационные матрицы разброса между классами и внутри классов.
3. Далее находятся собственные вектора и значения для линейного дискриминанта Фишера, который определяется отношением матриц из второго шага.

4. Собственные вектора сортируются в порядке убывания в соответствии с собственными значениями и называются дискриминантными векторами, с помощью которых рассчитываются веса модели.

5. Создаётся дискриминантное подпространство — линейная комбинация исходных признаков и дискриминантных векторов.

6. Спрогнозированные классы являются максимальной оценкой линейной комбинации тестовой выборки и весов + смещение.

На рисунке 1 представлена схема алгоритма LDA.

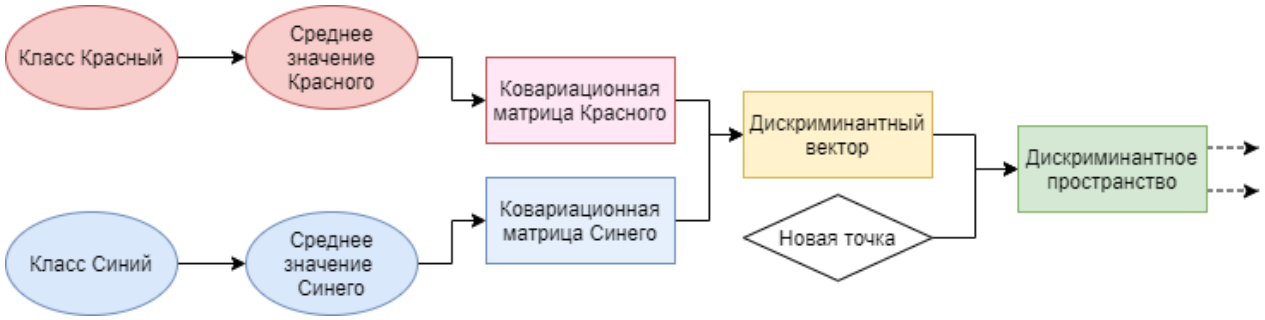

Рисунок 1 – Принцип работы LDA

Алгоритм K-ближайших соседей (**KNN**) — это метод классификации и регрессии, основанный на предположении компактности, которое гласит, что объекты, находящиеся рядом друг с другом в пространстве признаков, имеют похожие значения целевой переменной или относятся к одному классу.

Принцип работы **KNN**:

1. Сначала определяется расстояние между тестовым образцом и всеми обучающими выборками.

2. Затем выбираются k ближайших образцов (соседей), где число k задано заранее.

3. Результатом прогноза среди выбранных k ближайших образцов будет мода в случае классификации и среднее арифметическое в случае регрессии.

4. Предыдущие этапы повторяются для всех тестовых выборок.

На рисунке 2 представлена схема алгоритма KNN.

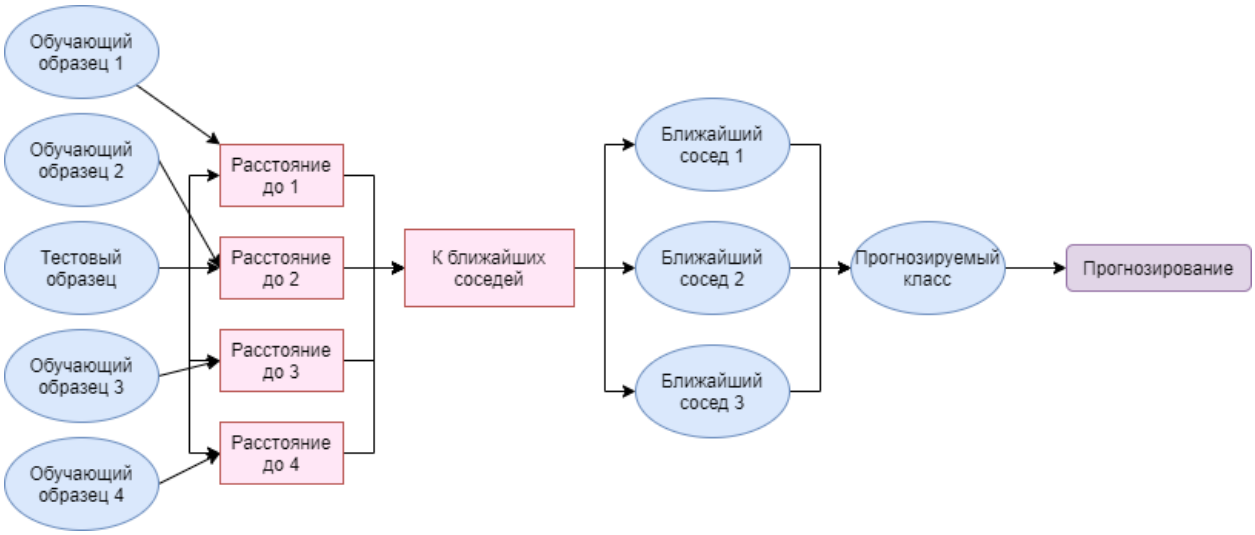

Рисунок 2 - Принцип работы KNN

Наивный байесовский классификатор — это вероятностный метод классификации, основанный на теореме Байеса и предполагающий строгую независимость признаков друг от друга в рамках определённого класса. Это существенно упрощает процесс классификации благодаря оценке одномерных вероятностных распределений вместо одного многомерного.

Принцип работы **наивного байесовского классификатора**:

1. Рассчитываются априорные вероятности классов.
2. Вычисляются средние и стандартные отклонения признаков по классам.
3. Определяется вероятностная плотность тестовых признаков по распределению Гаусса на основе полученных отклонений признаков по классам.
4. Апостериорные вероятности рассчитываются как произведение априорных вероятностей классов и вероятностных плотностей тестовых признаков.
5. Итоговый прогноз — класс с максимальной апостериорной вероятностью.

На рисунке 3 представлена схема алгоритма.

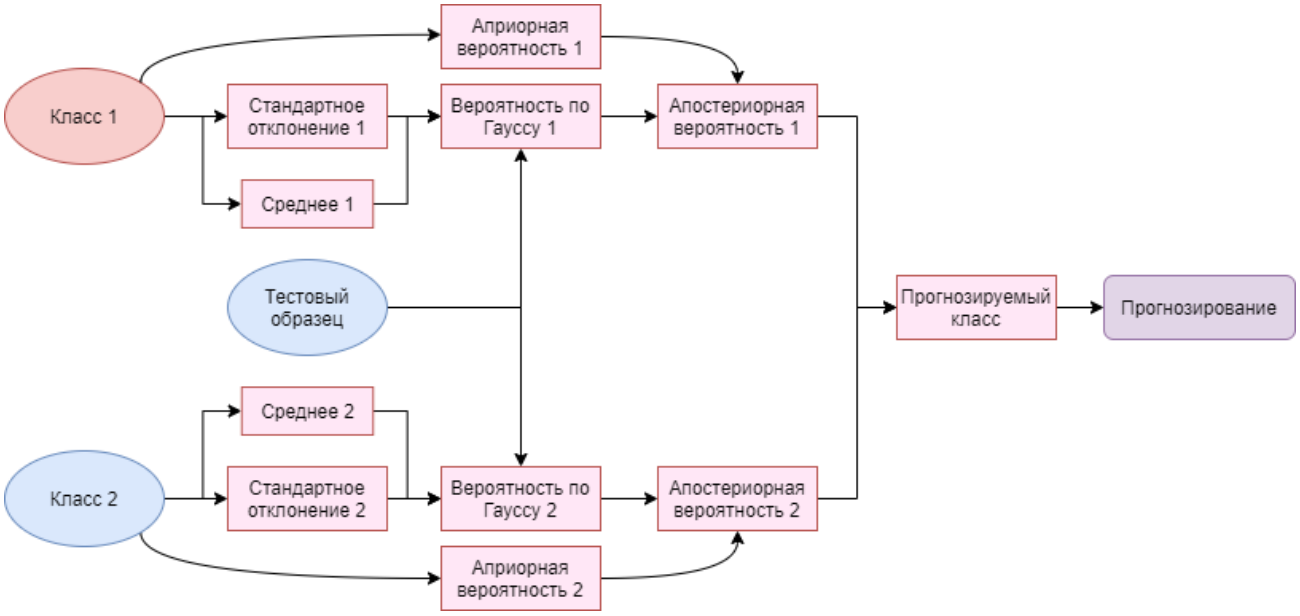

Рисунок 3 - Алгоритм наивного Байеса

Стандартизация (Standardization) — это процесс преобразования данных таким образом, чтобы они имели нулевое среднее значение и единичное стандартное отклонение.

Стандартизация позволяет добиться следующих вещей:

– устранить влияние масштаба признаков: к примеру в наборе данных Wine признаки могут иметь разный масштаб (например, одна переменная измеряется в миллиграммах, а другая - в процентах). Не стандартизированные данные могут привести к тому, что алгоритмы машинного обучения будут отдавать предпочтение признакам с большим масштабом, а не признакам с более важным содержанием;

– улучшить производительность алгоритмов: многие алгоритмы машинного обучения, особенно те, которые используют расстояние между точками данных (например, kNN), работают лучше, когда данные стандартизированы.

Обучающая выборка – выборка, на которой модель обучается. Набор данных, который мы передаем нашей модели, чтобы изучить потенциальные закономерности и отношения.

Валидационная выборка — это набор данных, который используется в процессе разработки модели машинного обучения для подбора оптимального

набора гиперпараметров. Проверяемая выборка является независимым набором записей данных, используемым в целях отслеживания ошибок в ходе обучения для предотвращения его чрезмерности.

Тестовая выборка – выборка для тестирования полученной модели, не участвуют в процессе обучения. Полученный, на тестовой выборке, результат мы сравниваем с фактическими значениями и исходя из этого, оцениваем точность нашей модели.

Библиотеки, используемые в блокноте colab и их описание:

- sklearn.datasets;
- pandas: библиотека для анализа данных и их предобработки;
- matplotlib.pyplot и seaborn: библиотеки для визуализации данных;
- train_test_split: функция для разделения данных на обучающую и тестовую выборки;
- StandardScaler: библиотека для стандартизации данных;
- LinearDiscriminantAnalysis, KNeighborsClassifier, GaussianNB: библиотеки для LDA, KNN, Наивного Байеса;
- accuracy_score, classification_report, confusion_matrix: библиотеки для оценки моделей.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также с представленным примером в Google Colab.
2. Выполнить ход работы.
3. Выполнить индивидуальное задание.
4. Оценить полученные результаты.
5. Ответить на контрольные вопросы.

1 Ход работы

1.1 Загрузка данных

Для начала работы вам необходимо подключить набор данных Wine из sklearn. Он содержит информацию о винах, такую как сорт винограда, регион происхождения, год урожая, оценка качества и другие характеристики.

Далее выполните загрузку данных и выведите информацию о наборе данных. Затем создайте DataFrame – двумерную структуру данных, используемую в библиотеке Pandas для работы с табличными данными. Также просмотрите первые несколько строк созданного *df*, чтобы просмотреть его структуру и содержимое. Для этого используйте следующие строчки кода:

```
from sklearn.datasets import load_wine # Подключение набора данных
wine = load_wine() # Загрузка данных
print(wine.DESCR)   # Выводим информацию набора данных
print(wine.feature_names)  # Названия признаков
# Создание DataFrame
df = pd.DataFrame(data=wine.data, columns=wine.feature_names)
df['Class'] = wine.target
df.head()
```

1.2 Предобработка данных

Для работы с набором данных выполните проверку на пропущенные значения. Затем убедитесь, что все признаки имеют числовой тип данных.

```
print(df.isnull().sum())
print(df.dtypes)
```

1.3 Работа с моделью

Теперь выберите признаки, по которым будет происходить обучение модели. *X* – набор переменных, которые используются для предсказания или классификации целевой переменной *Y*. Целевая переменная – это переменная, которая описывает результат процесса.

Это можно выполнить, используя следующий код:

```
selected_features = ['alcohol', 'malic_acid', 'ash'] # Выбор признаков
X = df[selected_features]
y = df['Class']
```

Теперь разделите данные на обучающую и тестовую выборки. Сделайте так, чтобы 40% данных были использованы для тестирования моделей. train_test_split - функция из библиотеки scikit-learn, которая выполняет разделение данных на обучающую и тестовую выборки.

```
test_size=0.4 #Размер тестовой выборки.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
random_state=42)
```

Выполните стандартизацию данных и обучите модели. Для этого примените готовые методы из библиотеки sklearn.

```
scaler = StandardScaler()        #Создается объект для стандартизации данных
X_train = scaler.fit_transform(X_train) #Стандартизация обучающих данных
X_test = scaler.transform(X_test)       #Стандартизация тестовых данных
lda = LinearDiscriminantAnalysis()      #Создание модели LDA
lda.fit(X_train, y_train)               #Обучение модели на обучающих данных
y_pred_lda = lda.predict(X_test)        #Предсказание на тестовых данных
knn = KNeighborsClassifier(n_neighbors=5)#Создание модели knn с 5 соседями
knn.fit(X_train, y_train)               #Обучение модели на обучающих данных
y_pred_knn = knn.predict(X_test)        #Предсказание на тестовых данных
nb = GaussianNB()
nb.fit(X_train, y_train)
y_pred_nb = nb.predict(X_test)
```

1.4 Результат модели

После того, как обучите модели, оцените их эффективность. Для этого используйте следующие метрики: точность (accuracy), полноту, F1-меру. На рисунках 1.1 - 1.2 представлены результаты обученной модели.

```
print('Линейный дискриминантный анализ (LDA):')
print(f'Точность: {accuracy_score(y_test, y_pred_lda)}')
print(classification_report(y_test, y_pred_lda))
plt.figure(figsize=(15, 5)) plt.subplot(1, 3, 1)
plt.plot(range(len(y_test)), y_test, label='Истинный класс', marker='o',
linestyle='-')
plt.plot(range(len(y_pred_lda)), y_pred_lda, label='Предсказанный LDA',
marker='x', linestyle='--')
plt.title('LDA')
plt.xlabel('Номер образца')
plt.ylabel('Класс вина')
plt.legend()
```

```
Линейный дискриминантный анализ (LDA):
Точность: 1.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        26
           1       1.00      1.00      1.00        27
           2       1.00      1.00      1.00        19

    accuracy                           1.00        72
   macro avg       1.00      1.00      1.00        72
weighted avg       1.00      1.00      1.00        72
```

Рисунок 1.1 – Результат метрик

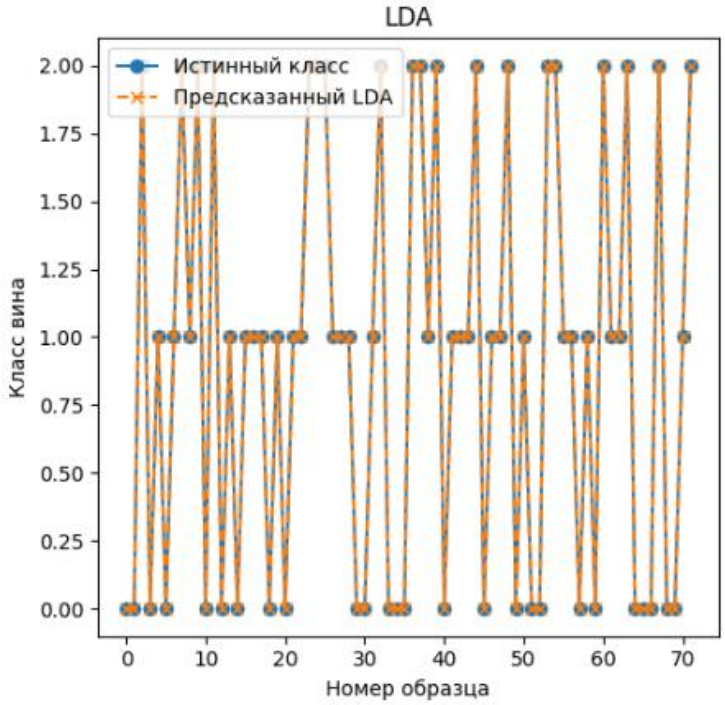

Рисунок 1.2 – График предсказаний

2 Индивидуальное задание

1. Выберете набор данных согласно варианту из таблицы 1.
2. Подготовьте датафрейм.
3. Выберите признаки, согласно варианту.
4. Разделите данные на обучающую и тестовую выборки согласно первому значению из таблицы согласно варианту.
5. Выполнит стандартизацию данных.
6. Выберите нужное количество соседей для kNN.
7. Обучите модели.
8. Проанализируйте полученные результаты при помощи метрик.
9. Постройте графики.
10. Повторите пункты 4 - 9, используя второе значение из таблицы для тестовой выборки.
11. Проанализируйте полученные результаты.

К примеру, если необходимо сделать 1-ый вариант, то нужно выбрать набор данных Iris, использовать длину лепестка и длину чашелистика как признаки. При разделении данных на тестовую и обучающую выборки, сначала нужно выполнить пункты 4 - 9, с размером тестовой выборки равной 30%, а затем с тестовой выборкой 50%. Количество соседей для kNN будет равняться трём.

Таблица 1 – Варианты индивидуального задания

Вариант	Набор данных	Признаки	Кол-во соседей	Размеры тест. выборки	
1	Iris	длина лепестка, длина чашелистика	3	30%	50%
2	Wine	алкоголь, яблочная кислота	5	30%	50%
3	Wine	флавоноиды, интенсивность цвета	4	20%	40%
4	Iris	длина лепестка, ширина лепестка	7	20%	40%
5	Wine	зола, пролин	3	15%	45%
6	Iris	ширина лепестка, ширина чашелистика	5	35%	50%
7	Iris	ширина лепестка, длина чашелистика	9	45%	15%

8	Wine	яблочная кислота, пролин	6	40%	10%
9	Wine	нефлаваноидные фенолы, насыщенность цвета	5	20%	35%
10	Wine	общее содержание фенолов, od280/od315 из расширенных вин	3	45%	15%
11	Iris	ширина лепестка, длина чашелистика	4	35%	60%
12	Iris	длина лепестка, длина чашелистика	6	10%	25%

Контрольные вопросы

1. Как выбирается оптимальное значение k для метода k-ближайших соседей (kNN)?
2. Как изменяется точность классификации LDA при изменении размеров выборки в вашем варианте?
3. Каковы преимущества и недостатки использования LDA?
4. Каковы преимущества и недостатки использования kNN?
5. Каковы преимущества и недостатки использования наивного байесовского классификатора?
6. Зачем нужна стандартизация данных?
7. Для чего нужно оценивать производительность модели?
8. В чем состоит предположение о независимости признаков в наивном байесовском классификаторе?

ЛАБОРАТОРНАЯ РАБОТА №3

Кластерный анализ с применением алгоритма k-means

Целью работы является изучение и практическое применение кластерного анализа с применением алгоритма k-means.

Краткие теоретические сведения

Метод k-средних (k-Means Clustering) – это очень известный и мощный алгоритм обучения без учителя (Unsupervised Learning), который группирует похожие элементы в *k* кластеров. Он используется для решения многих сложных задач машинного обучения (ML).

Кластеризация – контролируемый метод машинного обучения, используемый для прогнозирования значения зависимой переменной для новых, невидимых данных. Он моделирует взаимосвязь между входными характеристиками и целевой переменной, позволяя оценивать или предсказывать числовые значения.

Иерархическая кластеризация — это совокупность алгоритмов упорядочивания данных, направленных на создание иерархии (дерева) вложенных кластеров.

Задача регрессионного анализа работает, если выходной переменной является реальное или непрерывное значение, такое как «зарплата» или «вес». Можно использовать множество различных моделей, самой простой является линейная регрессия. Он пытается сопоставить данные с наилучшей гиперплоскостью, проходящей через точки.

Алгоритм очень чувствителен к масштабу признаков. В связи с этим нормализация данных (feature scaling) приобретает особое значение. Так как при формировании кластеров мы измеряем расстояние (в частности, Евклидово расстояние), то признаки с большим масштабом будут иметь больший вес.

Основные концепции:

- **Кластер**: это группа объектов, которые имеют схожие характеристики и расположены близко друг к другу в многомерном пространстве.
- **Центроид**: это центральная точка кластера, которая определяется как среднее значение всех точек, входящих в кластер.

Алгоритм k-means:

- **Выбор количества кластеров (k):** Заранее определяется количество кластеров, на которое будут разделены данные.
- **Инициализация центроидов:** Алгоритм случайным образом выбирает k точек данных в качестве начальных центроидов.
- **Присвоение точек кластерам:** Каждая точка данных присваивается ближайшему центроиду.
- **Обновление центроидов:** Положение каждого центроида обновляется как среднее значение всех точек, присвоенных данному кластеру.
- **Повторение:** Шаги 3 и 4 повторяются до тех пор, пока центроиды не перестанут существенно изменяться.

Преимущества k-means:

- Простота реализации и быстрота работы.
- Хорошо работает на больших наборах данных.

Недостатки k-means:

- Требует заранее задать количество кластеров.
- Чувствителен к выбросам и шуму в данных.
- Оценка качества кластеризации:

Для оценки качества кластеризации используются такие методы, как:

Метод локтя: используется для определения оптимального числа кластеров.

Силуэтный коэффициент: измеряет, насколько объект хорошо совпадает со своим кластером по сравнению с другими кластерами.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также представленными примерами в Google Colab.
2. Выполнить ход работы.
3. Выполнить индивидуальное задание.
4. Оценить полученные результаты.
5. Ответить на контрольные вопросы.

1 Ход работы

1.1 Загрузка данных

Перед началом работы необходимо скачать набор данных: https://www.kaggle.com/datasets/yasserh/wine-quality-dataset.

Этот набор данных относится к красным вариантам португальского вина «Винью Верде». Набор данных описывает количество различных химических веществ, присутствующих в вине, и их влияние на его качество.

После того как набор данных будет скачан, его необходимо загрузить в файловое пространство Colab.

1.2 Предварительный анализ данных

Постройте гистограммы для визуального анализа распределения переменных качества *(quality)* и летучей кислотности (*volatile acidity)*. Этот шаг помогает лучше понять диапазон значений и распределение качества вина. Гистограмма качества показывает, как распределены значения качества в выборке. Гистограмма летучей кислотности демонстрирует, какие значения чаще всего встречаются. Полученные гистограммы представлены на рисунке 1.1.

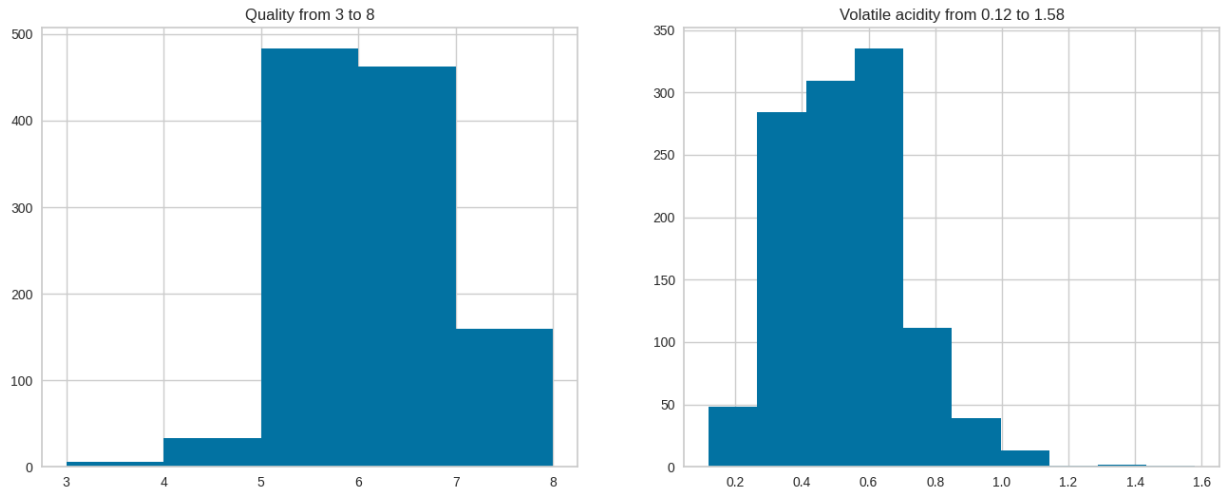

Рисунок 1.1 – Гистограмма качества и летучей кислотности

Для построения графиков, сначала выберите шесть признаков для построения парных графиков: «density», «pH», «sulphates», «alcohol», «citric acid» и «quality» (рисунок 1.2).

Для их создания используется функция sns.pairplot(), которая строит графики для сравнения всех выбранных признаков между собой. При этом цвет точек зависит от переменной quality, что позволяет визуально оценить, как качество вина соотносится с другими признаками.

Парные графики позволяют выявить возможные взаимосвязи между переменными и определить, как одна переменная изменяется в зависимости от другой. Например, можно оценить, есть ли связь между содержанием алкоголя и качеством вина или между кислотностью и плотностью.

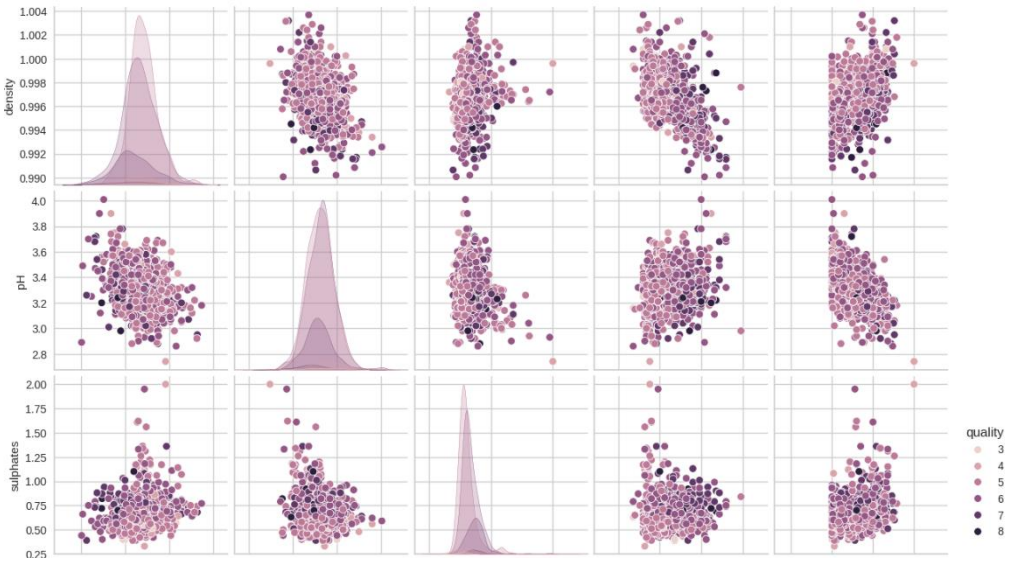

Рисунок 1.2 – Парные графики

Выведите корреляционную матрицу для оценки связи между параметрами вина (рисунок 1.3).

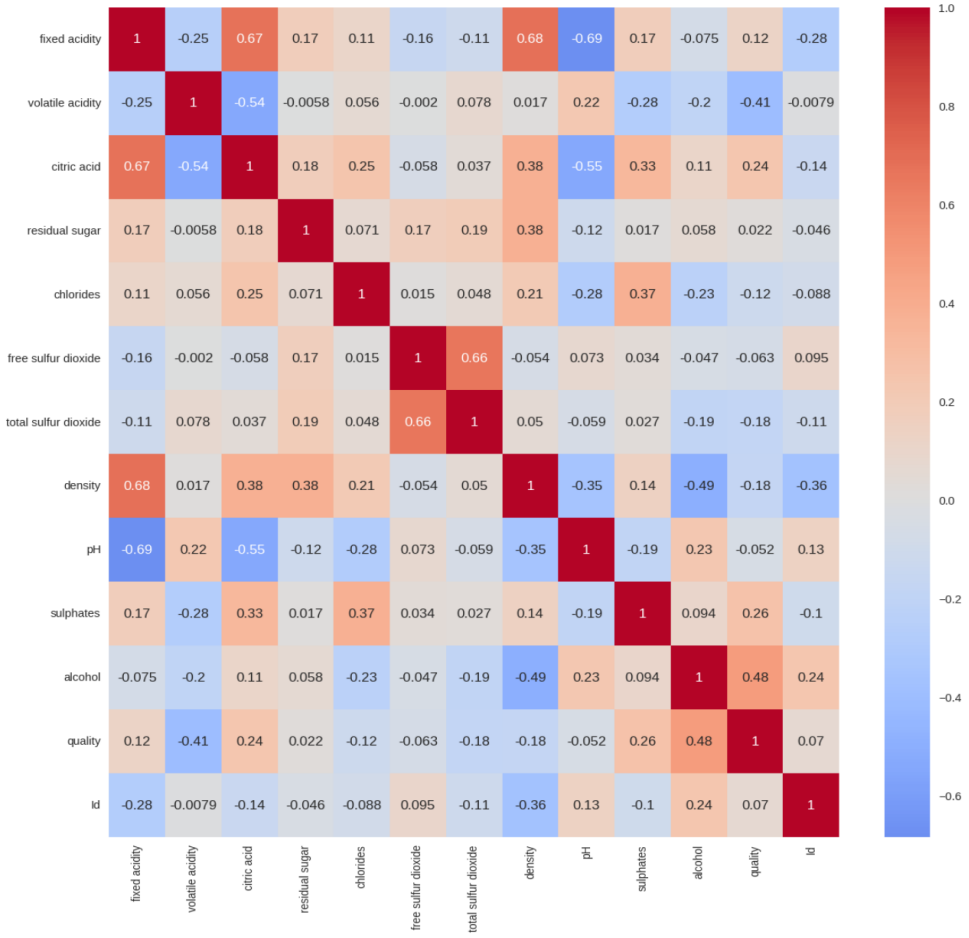

Рисунок 1.3 – Корреляционная матрица

1.3 Кластеризация

Проведите иерархическую кластеризацию с использованием метода «ward», после чего постройте дендрограмму (рисунок 1.4).

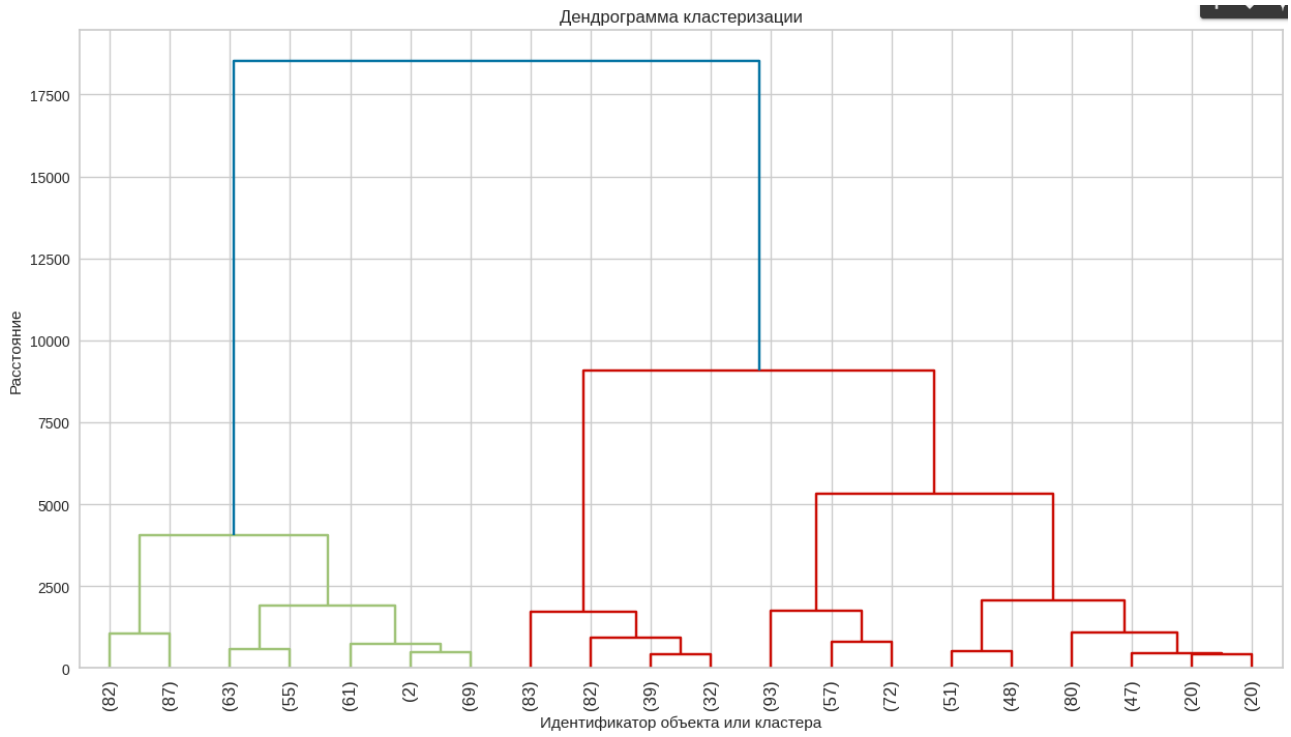

Рисунок 1.4 – Визуализация данных в 3D

После создания кластеров важно определить, какие элементы принадлежат каждому кластеру. Для этого можно использовать функцию fcluster(), которая создаёт массив с номерами кластеров для каждого элемента, сохраняя соответствие с исходными индексами. Это позволяет понять, к какому кластеру относится каждый элемент.

```
# Иерархическая кластеризация
Z = linkage(wine_data, "ward")
disctances_clusters = fcluster(Z, 3, criterion="distance")
max_clusters = fcluster(Z, 3, criterion="maxclust")
depth_clusters = fcluster(Z, 3, depth=5)
```

Функция fcluster() может работать по разным критериям: она может учитывать расстояния между элементами, задать фиксированное количество

кластеров, установить минимальное количество кластеров или анализировать определённую глубину иерархических уровней.

В задаче кластеризации существуют множество факторов, на основании которых производится окончательная классификация данных. Эти факторы в основном представляют собой атрибуты или характеристики элементов, которые используются для определения принадлежности к кластерам.

Чем больше признаков используется в модели, тем сложнее с ними работать. Многие из них могут быть коррелированными и избыточными, что снижает эффективность модели. Поэтому перед использованием классификатора часто проводят уменьшение размерности данных, чтобы сократить количество признаков и улучшить качество работы модели.

Снижение размерности — это процесс уменьшения количества рассматриваемых случайных величин путем получения набора основных переменных.

Анализ главных компонентов (PCA) — это метод уменьшения размерности наборов данных, который теряет наименьшее количество информации.

```
# Снижение размерности с помощью PCA
pca = PCA(n_components=3)
pca.fit(wine_data)
PCA_ds = pd.DataFrame(pca.transform(wine_data), columns=(["col1","col2",
"col3"]))
PCA_ds.describe().T
```

Также визуализируйте данные в уменьшенном трехмерном пространстве (рисунок 1.5).

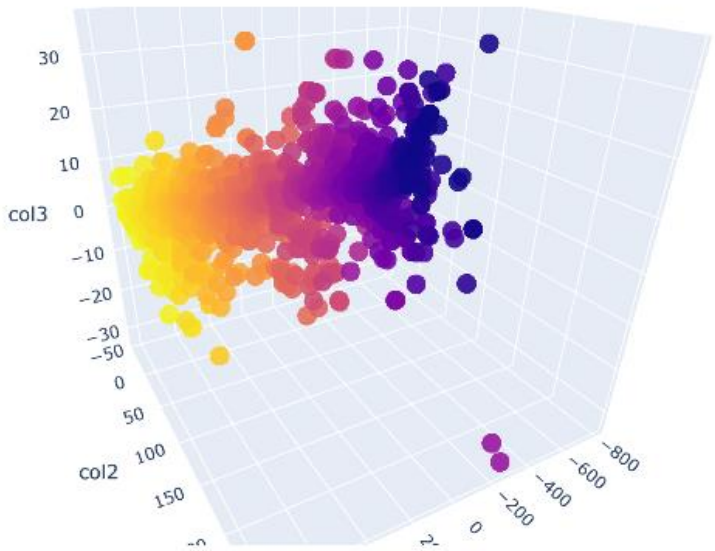

Рисунок 1.5 – Визуализация данных в 3D

Далее используется метод локтя для определения числа кластеров. С помощью KElbowVisualizer определяется оптимальное количество кластеров для алгоритма k-means (рисунок 1.6). По графику можно обнаружить, что оптимальное количество кластеров, исходя из данных, равно 4.

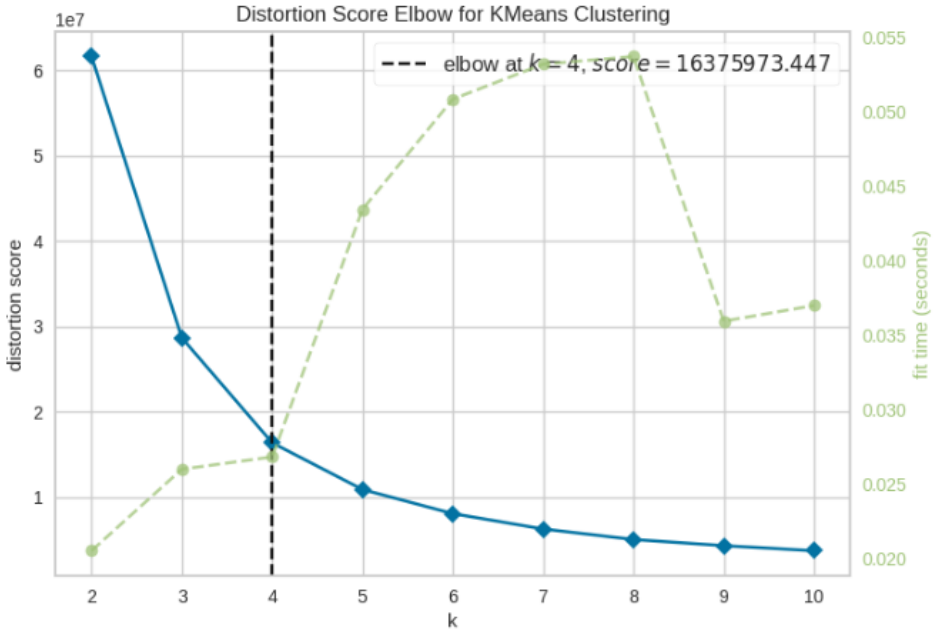

Рисунок 1.6 – Метод локтя

Далее выполните кластеризацию с использованием KMeans на основе оптимального количества кластеров.

Оцените внутрикластерную дисперсию (inertia) и качество кластеризации с помощью коэффициента силуэта, а также рассчитайте средние значения признаков для каждого кластера.

- Inertia (внутрикластерная дисперсия)
- Silhouette Score (коэффициент силуэта)

```
# Кластеризация с использованием KMeans
kmeans = KMeans(n_clusters=4, random_state=42)
wine_data['KMeans_Clusters'] = kmeans.fit_predict(PCA_ds)
# Получение координат центроидов
centroids = kmeans.cluster_centers_
# Оценка внутрикластерной вариабельности и качества кластеризации
inertia = kmeans.inertia_
silhouette_score = metrics.silhouette_score(PCA_ds,
wine_data['KMeans_Clusters'])
print("Inertia (внутрикластерная вариабельность):", inertia)
print("Silhouette Score (оценка качества кластеризации):", silhouette_score)
# Анализ характеристик объектов внутри каждого кластера
cluster_analysis = wine_data.groupby('KMeans_Clusters').mean()
print("Средние значения характеристик для каждого кластера:")
print(cluster_analysis)
```

Низкое значение вариабельности указывает на хорошую внутрикластерную плотность.

Коэффициент силуэта оценивает качество кластеризации, измеряя, насколько объект ближе к своему кластеру, чем к соседним. Значение варьируется от -1 до 1: чем ближе к 1, тем лучше объект соответствует своему кластеру. Значение выше 0.5 обычно указывает на чёткое разделение кластеров, так как расстояние до своего кластера значительно меньше, чем до соседнего.

Далее для удобства восприятия визуализируйте кластеры и центроиды в 3D-пространстве, а также постройте гистограмму распределения объектов по кластерам (рисунки 1.7 – 1.8).

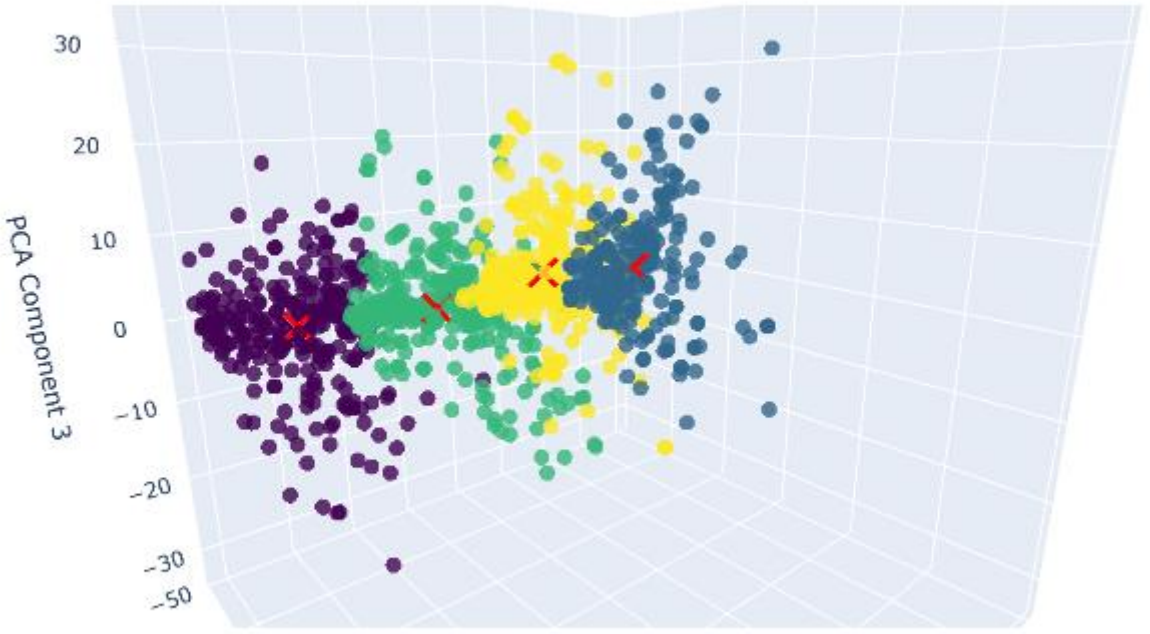

Рисунок 1.7 – Визуализация кластеров и центроидов в 3D-пространстве

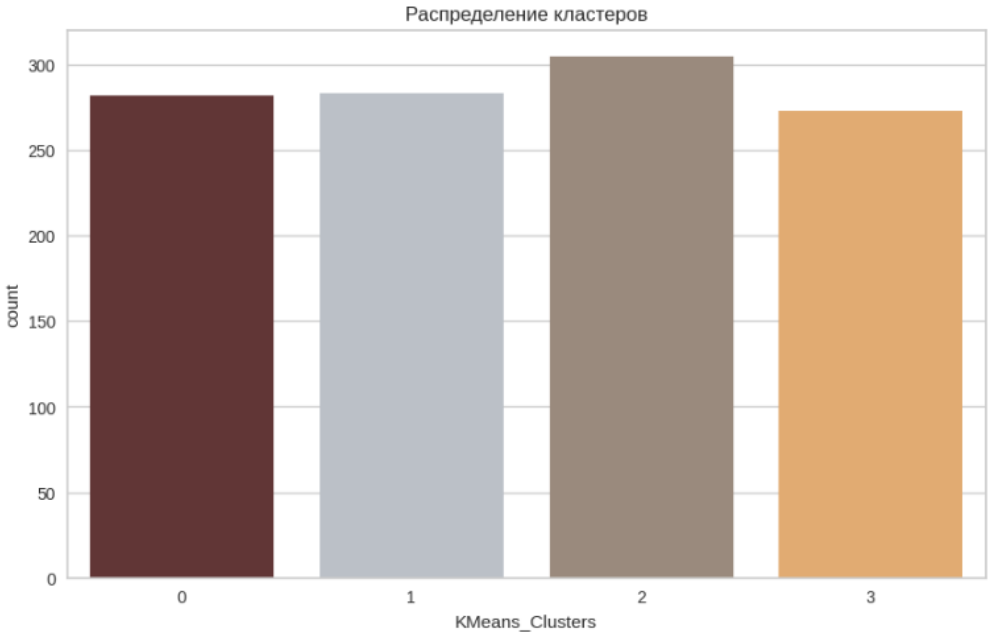

Рисунок 1.8 – Гистограмма распределения объектов по кластерам

3D-визуализация позволяет увидеть распределение кластеров в пространстве главных компонент. Гистограмма показывает, сколько объектов попало в каждый кластер.

Создайте диаграмму рассеяния (scatter plot) для переменных «residual sugar» и «pH», где точки окрашены в зависимости от кластера, к которому они принадлежат. Этот график наглядно демонстрирует, как кластеры разделены в пространстве по параметрам «residual sugar» и «pH». Интерфейс позволяет пользователю взаимодействовать с графиком, что облегчает анализ и понимание результатов кластеризации (рисунок 1.9).

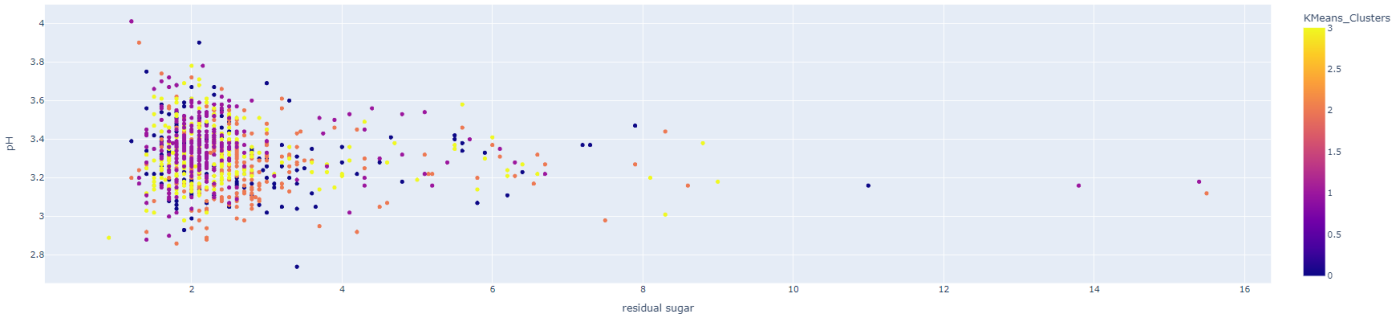

Рисунок 1.9 – Дополнительная визуализация с использованием Plotly

2 Индивидуальное задание

После выполнения основной работы необходимо выполнить задания по вариантам:

1. Загрузите набор данных в соответствии со своим вариантом.
2. Выполните предварительный анализ данных:
   а. Определите количество уникальных классов.
   б. Проверьте наличие пропущенных значений.
3. Примените метод главных компонент (PCA) для сокращения размерности данных.
4. Выполните кластеризацию методом KMeans с количеством кластеров от 2 до 5.
5. Визуализируйте полученные кластеры на графике и сравните их с реальными классами.
6. Определите оптимальное количество кластеров:

а. Используйте метод локтя.

б. Рассчитайте и проанализируйте оценку коэффициента силуэта.

7. Подготовьте отчет, включив визуализацию и описание полученных кластеров.

Наборы данных для работы представлены в таблице 2.1.

Таблица 2.1 – Наборы данных по вариантам

Вариант	Набор данных
1	https://www.kaggle.com/datasets/maksymshkliarevskyi/plantstat-package-statistics-and-automl/data
2	https://www.kaggle.com/datasets/ramontanoeiro/big-five-personality-test-removed-nan-and-0/data
3	https://www.kaggle.com/datasets/krishnaraj30/mall-visiting-customer-data
4	https://www.kaggle.com/datasets/rohan0301/unsupervised-learning-on-country-data
5	https://www.kaggle.com/datasets/vishakhdapat/customer-segmentation-clustering
6	https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python
7	https://www.kaggle.com/datasets/ulrikthygepedersen/online-retail-dataset/data
8	https://www.kaggle.com/datasets/youssefaboelwafa/clustering-penguins-species

Контрольные вопросы

1. Что такое кластеризация и где она применяется?
2. Как работает алгоритм k-means?
3. Что такое центроид кластера?
4. Почему важно нормализовать данные перед кластеризацией?
5. Как можно определить оптимальное количество кластеров?
6. Что измеряет силуэтный коэффициент (Silhouette Score)?
7. Как выбросы в данных могут повлиять на результаты кластеризации?

ЛАБОРАТОРНАЯ РАБОТА № 4

Разработка модели дерева решений и случайного леса для анализа данных

Целью лабораторной работы является практическое применение дерева решений и случайного леса для анализа данных.

Краткие теоретические сведения

Деревья решений — это метод, позволяющий компьютерам принимать решения, основываясь на наборе правил, подобно тому, как человек принимает решения. Эти правила организованы в древовидную структуру, где каждый узел дерева представляет собой вопрос или условие, а ветви указывают на возможные ответы или действия.

Узел дерева решений — это точка, в которой происходит разделение данных на основе определённого атрибута.

Ветви дерева решений — это направления, по которым данные следуют после разделения в узле.

Лист дерева решений — это конечный узел дерева решений, который содержит предсказание или классификацию. Листья являются результатом процесса обучения и представляют собой выводы, сделанные на основе входных данных.

На рисунке 1 представлен пример дерева решений.

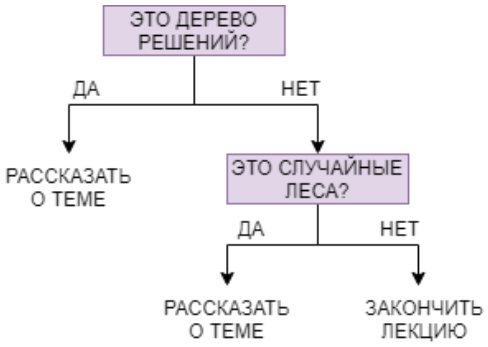

Рисунок 1 – Пример дерева решений

Случайный лес (Random Forest) – это мощный алгоритм машинного обучения, используемый для решения задач классификации и регрессии. Он основан на идее ансамбля деревьев решений, где каждое дерево строится на случайной выборке данных и случайном подмножестве признаков. Это позволяет уменьшить переобучение и повысить точность модели.

Основные принципы работы случайного леса включают:

– выбор подмножества данных: для каждого дерева решений случайным образом выбирается подмножество обучающих данных (bootstrap sample);

– выбор подмножества признаков: также случайным образом выбирается подмножество признаков, на основе которых будет строиться дерево;

– построение деревьев решений: на основе выбранных подмножеств данных и признаков строятся отдельные деревья решений;

– голосование: после построения всех деревьев для каждого объекта данных проводится голосование, и наиболее популярный класс становится предсказанным классом для задачи классификации или среднее значение результатов деревьев для задачи регрессии.

Обучающая выборка – выборка, на которой модель обучается. Набор данных, который мы передаем нашей модели, чтобы изучить потенциальные закономерности и отношения.

Валидационная выборка — это набор данных, который используется в процессе разработки модели машинного обучения для подбора оптимального набора гиперпараметров. Проверяемая выборка является независимым набором записей данных, используемым в целях отслеживания ошибок в ходе обучения для предотвращения его чрезмерности.

Тестовая выборка – выборка для тестирования полученной модели, не участвуют в процессе обучения. Полученный, на тестовой выборке, результат мы сравниваем с фактическими значениями и исходя из этого, оцениваем точность нашей модели.

Матрица ошибок используется для визуализации результатов классификации и показывает, как модель предсказывает классы по сравнению с фактическими классами.

Как читать матрицу ошибок:

– оси: Ось X (горизонтальная) представляет **предсказанные** классы. Ось Y (вертикальная) представляет **фактические** классы;

– ячейки: Каждая ячейка в матрице показывает количество примеров, которые были **фактически** отнесены к определенному классу (ось Y) и **предсказаны** как принадлежащие к другому классу (ось X);

– диагональ: Диагональ матрицы показывает количество правильно классифицированных примеров. Чем больше значение на диагонали, тем точнее модель.

На рисунке 2 представлена схема матрицы ошибок.

Рисунок 3 – Схема матрицы ошибок

Обозначения:

– TP (True Positive): Верно классифицированные как злокачественные;

– FN (False Negative): Неправильно классифицированные как доброкачественные, хотя они злокачественные (ошибка, которая может привести к задержке лечения);

– FP (False Positive): Неправильно классифицированные как злокачественные, хотя они доброкачественные (ошибка, которая может привести к ненужным тревогам);

– TN (True Negative): Верно классифицированные как доброкачественные.

Интерпретация матрицы ошибок:

– высокое значение TP: Модель хорошо классифицирует злокачественные опухоли;

– высокое значение TN: Модель хорошо классифицирует доброкачественные опухоли;

– высокое значение FN: Модель часто пропускает злокачественные опухоли (ошибка, которую нужно минимизировать);

– высокое значение FP: Модель часто ошибочно классифицирует доброкачественные опухоли как злокачественные (ошибка, которую нужно минимизировать).

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также с представленным примером в Google Colab.
2. Выполнить ход работы.
3. Выполнить индивидуальное задание.
4. Ответить на контрольные вопросы.

1 Ход работы

Для начала вам необходимо загрузить набор данных Breast Cancer из sklearn и создайте DataFrame.

Нужно вывести:

- первые несколько строк;
- информацию о наборе данных.

```
# Загружаем набор данных Breast Cancer
cancer = load_breast_cancer()
data = pd.DataFrame(data=cancer.data, columns=cancer.feature_names)
data["target"] = cancer.target
print(cancer.DESCR) # Информация о наборе данных
print(cancer.feature_names)  # Названия признаков
data.head()
```

Убедитесь, что нет пропущенных значений. Проверка на пропущенные значения нужна для предотвращения ошибок, улучшения качества данных и прозрачности процесса. Также необходимо проверить типы данных признаков нашего набора данных.

Это можно выполнить с помощью команды:

```
print(data.isnull().sum())
print(data.dtypes)
```

Теперь выберите признаки, по которым будет происходить обучение модели. Ниже представлены строчки кода для использования всех признаков.

```
features = cancer.feature_names  # Используем все признаки набор данных
target = "target"
```

Для обучения и тестирования модели нужно разделить данные на две части. Переменна test_size отвечает за размер тестовой выборки.

```
X_train, X_test, y_train, y_test = train_test_split(
    data[features], data[target], test_size=0.2, random_state=42)
```

Затем выполните обучение модели дерева решений и модели случайного леса. Для этого можно использовать готовые функции библиотеки sklearn. Ниже представлен пример реализации обучения моделей.

```
decision_tree_model = DecisionTreeClassifier() # --- Дерево решений ---
decision_tree_model.fit(X_train, y_train)
y_pred_dt = decision_tree_model.predict(X_test)
random_forest_model = RandomForestClassifier() # --- Случайный лес ---
random_forest_model.fit(X_train, y_train)
y_pred_rf = random_forest_model.predict(X_test)
```

После обучения моделей, их необходимо проверить. Для этого используйте функцию accuracy_score. Она вычисляется как отношение правильно предсказанных значений к общему количеству значений. Также используйте отчет о классификации, включающий в себя метрики precision, recall, f1-score. Используйте матрицы ошибок для визуализации работы модели.

Для того чтобы проверить модель дерева решений используйте код, представленный ниже. Для случайного леса ориентируйтесь на информацию в блокноте.

```
accuracy_dt = accuracy_score(y_test, y_pred_dt)
print(f"Точность дерева решений: {accuracy_dt}")
print("Отчет о классификации для дерева решений:")
print(classification_report(y_test, y_pred_dt))
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
# Визуализация матрицы ошибок для дерева решений
plt.figure(figsize=(8, 6))
plt.imshow(conf_matrix_dt, cmap=plt.cm.Blues)
plt.title("Матрица ошибок для дерева решений")
plt.xticks([0, 1], ["Злокачественная", "Доброкачественная"])
plt.yticks([0, 1], ["Злокачественная", "Доброкачественная"])
plt.colorbar()
plt.xlabel("Предсказанный класс")
plt.ylabel("Фактический класс")
for i in range(2):
    for j in range(2):
        plt.text(j, i, conf_matrix_dt[i, j], ha="center", va="center",
color="white")
plt.show()
```

2 Задания на лабораторную работу

Измените модель согласно вашему варианту из таблицы 2.1, используя первое значение, для этого:

1. Разделите данные на обучающую и тестовую выборки.
2. Настройте глубину дерева.
3. Настройте минимальное количество образцов в листе.
4. Выберите критерий разделения.
5. Настройте количество признаков, рассматриваемых при каждом разбиении.
6. Выберите количество деревьев.
7. Выберите размер тестовой выборки.
8. Оцените эффективность модели.

Повторите для второго значения из таблицы вариантов 2.1 и сравните полученные результаты.

Таблица 2.1 – Варианты индивидуального задания

№В	Глуб. дерева	Мин. кол-во образцов в листе	Критерий разделения	Кол-во признаков, рассматрив. при разбиении (DT)	Кол-во деревьев (RF)	Размер тестовой выборки
1	3 / 5	5 / 20	gini / entropy	sqrt / auto	50 / 200	0.3 / 0.45
2	6 / 8	7 / 14	gini / entropy	log2 / auto	26 / 132	0.35 / 0.6
3	2 / 10	9 / 19	gini / entropy	log2 / sqrt	39 / 180	0.2 / 0/4
4	4 / 10	3 / 10	gini / entropy	auto / sqrt	50 / 100	0.4 / 0.6
5	3 / 6	10 / 20	gini / entropy	sqrt / log2	41 / 157	0.3 / 0.5
6	5 / 7	8 / 17	entropy / gini	auto / log2	52 / 68	0.25 / 0.55
7	3 / 5	5 / 14	entropy / gini	log2 / auto	26 / 134	0.2 / 0.6
8	6 / 9	7 / 19	entropy / gini	sqrt / auto	50 / 190	0.35 / 0.65
9	2 / 5	3 / 20	entropy / gini	log2 / sqrt	36 / 145	0.15 / 0.45
10	2 / 10	7 / 17	entropy / gini	auto / sqrt	20 / 200	0.2 / 0.4

То есть, если необходимо сделать 1-ый вариант, то модель должна быть изменена, согласно первому значению каждой колонки из таблицы 1: глубина дерева 3, мин. кол-во образцов в листе 5, критерий разделения gini, кол-во

признаков, рассматриваемых при каждом разбиении (для дерева решений) sqrt, кол-во деревьев (случайный лес) 50, размер тестовой выборки 0.3. Выполняется оценка эффективности измененной модели.

Затем необходимо повторить уже со вторым значением из таблицы: глубина дерева 5, мин. кол-во образцов в листе 20, критерий разделения entropy, кол-во признаков, рассматриваемых при каждом разбиении (для дерева решений) auto, кол-во деревьев (случайный лес) 200, размер тестовой выборки 0.3. Выполняется оценка эффективности измененной модели. В конце полученные результаты должны быть проанализированы.

Контрольные вопросы

1. Опишите, что такое дерево решений и как оно работает?
2. Объясните принцип работы случайного леса и его преимущества перед деревом решений.
3. В каких реальных задачах может быть использован метод случайного леса?
4. В каких реальных задачах может быть использована метод дерева решений?
5. Что такое матрица ошибок?
6. Какие есть недостатки у дерева решений?
7. Какие есть недостатки у случайного леса?
8. На что влияет изменение глубины деревьев в случайном лесу и как?

ЛАБОРАТОРНАЯ РАБОТА №5

Снижение размерности данных методом главных компонент и использование машины опорных векторов

Целью работы является изучение и практическое применение метода главных компонент (PCA), а также модели машины опорных векторов (SVM)

Краткие теоретические сведения

Машина опорных векторов (SVM) — это мощный алгоритм машинного обучения, который используется для классификации, регрессии и обнаружения аномалий. Главная цель SVM — определить оптимальную границу разделения, которая эффективно разделяет различные классы данных, обеспечивая точную классификацию и прогнозирование.

Метод главных компонент (Principal Component Analysis, PCA) — это метод снижения размерности данных, который используется для уменьшения числа переменных в модели, сохраняя как можно больше вариаций данных. Метод основан на поиске новых переменных (главных компонент), которые представляют собой линейные комбинации исходных переменных. Основные компоненты выбираются так, чтобы объяснить максимальную дисперсию данных.

Основные этапы PCA:

- **Центрирование данных**: Прежде чем выполнять PCA, данные центрируются (вычитание среднего значения).
- **Выбор главных компонент**: Находятся собственные векторы (направления) и собственные значения (дисперсии вдоль этих направлений) ковариационной матрицы данных. Первые несколько векторов с наибольшими собственными значениями определяют главные компоненты.

- **Проекция данных**: Данные проецируются на подпространство, образованное выбранными главными компонентами, что уменьшает размерность данных.

Метод PCA часто применяется для предварительной обработки данных перед использованием алгоритмов машинного обучения, таких как машины опорных векторов (SVM), чтобы уменьшить вычислительную сложность и предотвратить проблему переобучения.

Машина опорных векторов — это метод машинного обучения, используемый для решения задач классификации и регрессии. SVM пытается найти гиперплоскость в многомерном пространстве, которая максимально разделяет данные на классы. Основная идея алгоритма заключается в максимизации ширины разделяющей полосы, ограниченной опорными векторами (данными, наиболее близкими к разделяющей гиперплоскости).

Основные этапы работы SVM:

- **Выбор ядра**: Определение функции ядра (линейное, полиномиальное, гауссово и т.д.), которая преобразует данные в новое пространство.
- **Построение разделяющей гиперплоскости**: SVM строит гиперплоскость, которая максимально отделяет данные одного класса от данных другого.
- **Оптимизация параметров**: Использование различных параметров регуляризации и настройки гиперпараметров, таких как параметр C (наказание за ошибку классификации).

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями, а также представленными примерами в Google.Colab.
2. Выполнить индивидуальное задание.

3. Выполнить ход работы.
4. Оценить полученные результаты.
5. Ответить на контрольные вопросы.

1 Ход работы

1.1 Загрузка данных

Перед началом работы необходимо скачать набор данных: https://www.kaggle.com/datasets/uciml/iris.

Этот набор данных содержит 150 образцов трёх видов ирисов, по 50 экземпляров каждого. Для каждого цветка представлены четыре характеристики: длина и ширина чашелистиков (sepal), а также длина и ширина лепестков (petal).

После того как набор данных будет скачан, его необходимо загрузить в файловое пространство Colab.

1.2 Предварительный анализ данных

Сначала производится исследование данных с помощью визуализации, чтобы лучше понять их структуру и выявить возможные зависимости между признаками. Визуализация признаков в виде диаграмм рассеяния позволяет оценить, насколько хорошо данные разделяются по классам, и выбрать наиболее информативные признаки.

Полученные диаграммы рассеяния для различных пар признаков представлены на рисунке 1.1.

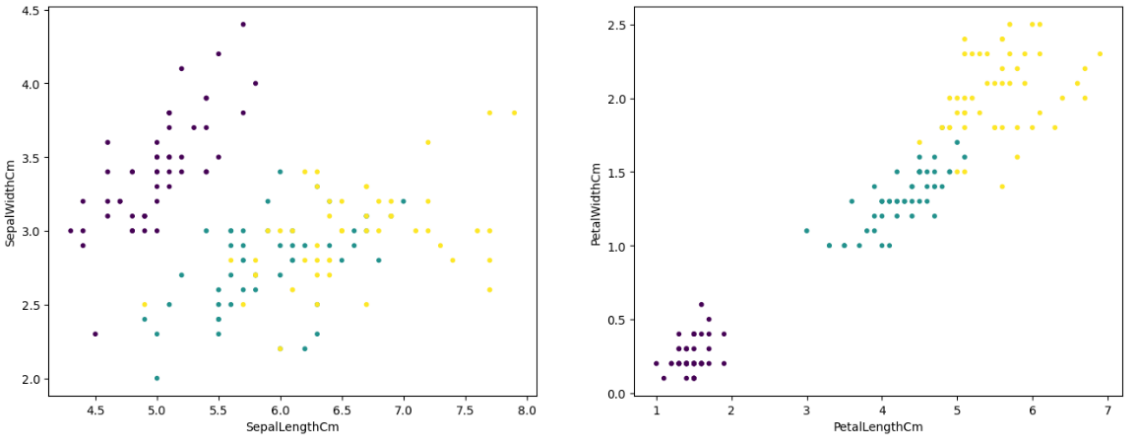

Рисунок 1.1 – Диаграммы рассеяния

Далее постройте матрицу парных графиков, по которым можно увидеть, что некоторые классы более четко разделяются по определенным признакам (таким как длина лепестка), чем по другим (рисунок 1.2).

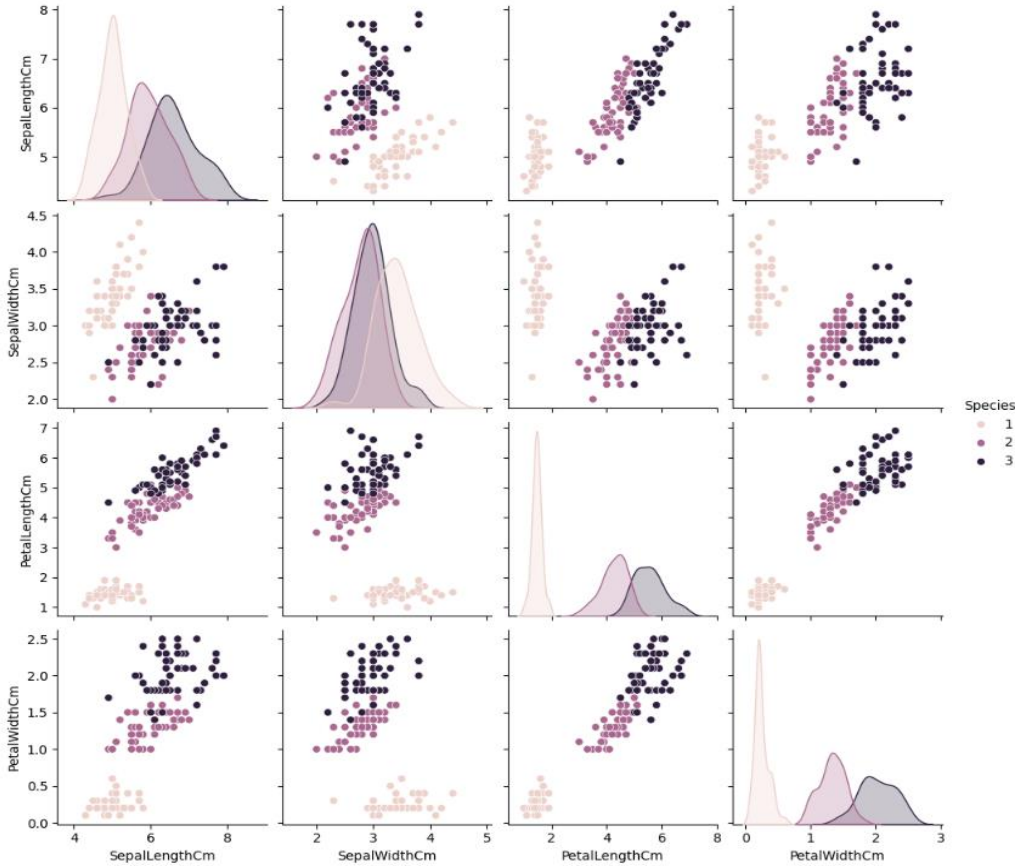

Рисунок 1.2 – Парные графики

Далее выведите тепловую корреляционную матрицу, по которой видно, что все признаки имеют сильную положительную или отрицательную корреляцию с целевым векторным видом (меткой), который будет классифицироваться (рисунок 1.3).

Рисунок 1.3 – Корреляционная матрица

Создайте гистограммы и коробчатые диаграммы для визуализации распределения признаков (рисунок 1.4).

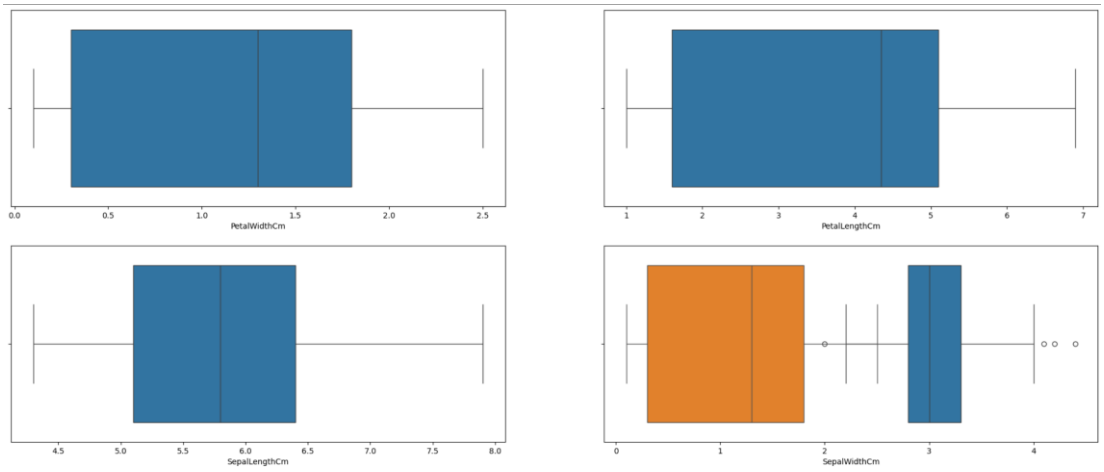

Рисунок 1.4 – Коробчатые диаграммы

Гистограммы показывают распределение данных для каждого признака, что позволяет оценить, какие признаки имеют нормальное распределение, а какие — нет. Например, если гистограмма одного из признаков сильно смещена в одну сторону, это может повлиять на выбор методов обработки данных.

Коробчатые диаграммы (boxplot) помогают обнаружить выбросы (аномальные значения) в данных. Выбросы могут негативно повлиять на обучение модели, поэтому их важно идентифицировать и, при необходимости, удалить.

Выполните очистку данных, этот этап включает в себя удаление выбросов и нормализацию данных для подготовки их к моделированию. Используйте Z-оценку для определения выбросов и их удаления.

```
from scipy import stats

z = np.abs(stats.zscore(X))
zee = (np.where(z > 2.5))[1]
X = X[(z <= 2.5)]
```

Удаление выбросов помогает уменьшить влияние аномальных данных, которые могут исказить результаты обучения модели. Удаление данных с Z-оценкой больше 2.5 означает удаление всех значений, которые сильно отклоняются от среднего, что помогает стабилизировать обучение модели.

Примените StandardScaler для нормализации данных, чтобы все признаки имели одинаковый масштаб, что улучшает производительность модели.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

1.3 Снижение размерности методом PCA

Примените метод главных компонент (PCA) для снижения размерности данных. В данной лабораторной работе выберите три главные компоненты для сохранения максимальной возможной информации из исходного набора данных.

```
# Применение PCA
fig = plt.figure(1, figsize=(16, 9))
ax = Axes3D(fig, elev=-150, azim=110)
X_reduced = PCA(n_components=3).fit_transform(X_scaled)
```

Выведите интерактивную 3D-визуализацию после выполнения PCA с тремя компонентами (рисунок 1.5).

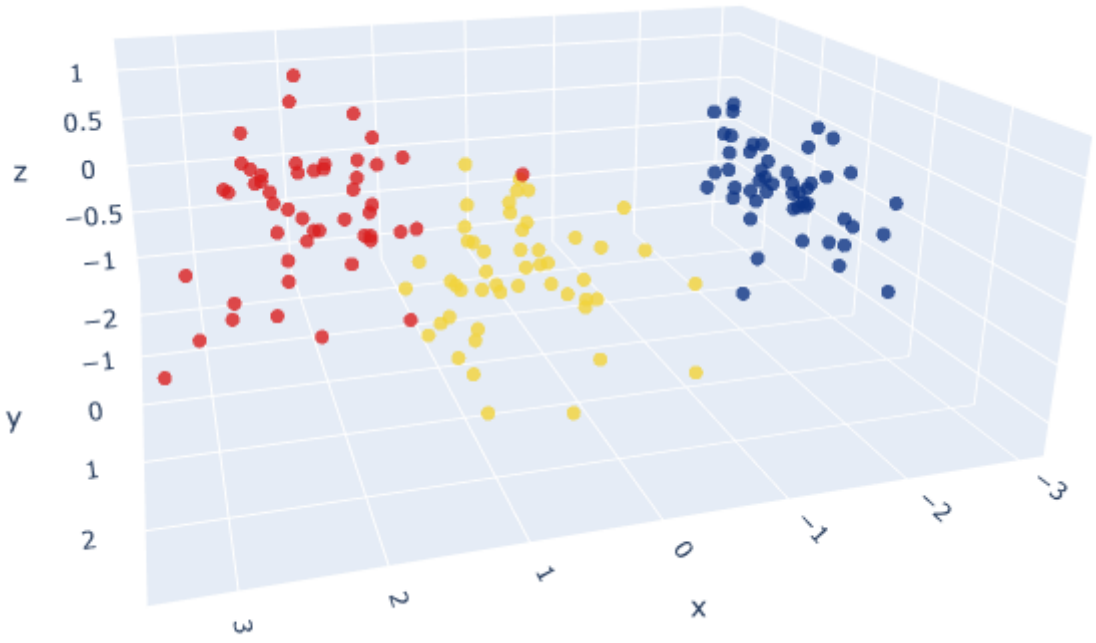

Рисунок 1.5 – Визуализация данных в 3D

1.4 Обучение модели SVM

Разделите данные на обучающие и тестовый наборы в пропорции 80/20. После чего обучите модель SVM с линейным ядром (LinearSVC). Линейное ядро хорошо подходит для задач, где классы можно линейно разделить.

```
from sklearn.svm import LinearSVC
clf = LinearSVC(penalty='l2', loss='squared_hinge',
                dual=True, tol=0.0001, C=100, multi_class='ovr',
                fit_intercept=True, intercept_scaling=1,
class_weight=None,verbose=0
                , random_state=0, max_iter=1000)
clf.fit(X_train,y_train)
```

Оптимизация гиперпараметров модели SVM проводится с использованием метода Grid Search. Используется сеточный поиск (GridSearchCV), чтобы найти наилучшие значения гиперпараметра *C*, которые обеспечат максимальную точность модели. Параметр *C* определяет, насколько строго модель будет

избегать ошибок классификации: большие значения *C* могут привести к переобучению модели.

```
# Определяем сетку значений параметра C для поиска
c = np.logspace(start=-15, stop=1000, base=1.02)

param_grid = {'C': c}

# Настройка Grid Search с использованием кросс-валидации
grid = GridSearchCV(clf, param_grid=param_grid, cv=3, n_jobs=-1,
scoring='accuracy')

grid.fit(X_train, y_train)
```

Затем обучите модель SVM с радиальным базисным ядром (RBF), чтобы обнаружить более сложные паттерны в данных.

```
from sklearn.svm import SVC
# Инициализация модели SVM с нелинейным (RBF) ядром
clf_SVC = SVC(C=100.0, kernel='rbf', degree=3, gamma='auto', coef0=0.0,
              shrinking=True, probability=False, tol=0.001, cache_size=200,
              class_weight=None, verbose=0, max_iter=-1,
              decision_function_shape="ovr", random_state=0)
# Обучение модели на обучающих данных
clf_SVC.fit(X_train, y_train)
```

Нелинейная модель SVM с RBF ядром лучше подходит для данных с более сложной структурой. Она может лучше выявлять и обобщать сложные закономерности, что часто приводит к повышению точности на тестовых данных.

2 Индивидуальное задание

После выполнения основной работы необходимо выполнить задания по вариантам.

Для выполнения заданий по вариантам необходимо скачать набор данных соответствующий варианту.

- Провести предобработку данных:
- Заполнить пропуски;
- Закодировать категориальные признаки;
- Применить PCA для снижения размерности признаков (в отчете предоставить максимально достигнутую точность);
- Обучить модель SVM (в отчете предоставить максимально достигнутую точность).

Наборы данных для работы по вариантам представлены в таблице 2.1.

Таблица 2.1 – Наборы данных по вариантам

Вариант	Набор данных
1	https://www.kaggle.com/datasets/henriqueyamahata/bank-marketing/data
2	https://www.kaggle.com/datasets/jahnveenarang/cvdcvd-vd/data
3	https://www.kaggle.com/datasets/samira1992/mobile-price-intermediate-dataset/data
4	https://www.kaggle.com/datasets/yasserh/wine-quality-dataset/data
5	https://www.kaggle.com/datasets/uciml/pima-indians-diabetes-database/data
6	https://www.kaggle.com/datasets/binovi/wholesale-customers-data-set
7	https://www.kaggle.com/datasets/uciml/adult-census-income/data
8	https://www.kaggle.com/datasets/andrewmvd/heart-failure-clinical-data/data

Контрольные вопросы

1. Что такое метод главных компонент (PCA) и для чего он используется?
2. Почему важно стандартизировать данные перед применением метода PCA?
3. Чем отличается линейное ядро SVM от радиального базисного (RBF) ядра? В каких случаях предпочтительнее использовать каждое из них?
4. Почему важно удалять выбросы из набора данных перед обучением модели?
5. Какую роль играют опорные векторы в SVM?
6. Какие задачи могут решаться с помощью метода главных компонент, помимо снижения размерности?

ЛАБОРАТОРНАЯ РАБОТА № 6

Практическое применение нейронных сетей в анализе данных

Цель работы: практическое применение многослойных перцептронов (MLP) для решения задачи бинарной классификации на примере данных о диабете.

Краткие теоретические сведения

Нейрон — это базовый элемент нейронной сети, который моделирует работу биологического нейрона. Он принимает входные данные, обрабатывает их с помощью активационной функции и выдает результат.

Слой в контексте нейронных сетей — это набор нейронов, организованных в определённую структуру, которые выполняют определённые вычисления над входными данными.

Нейронная сеть — это модель машинного обучения, вдохновленная структурой и функционированием биологических нейронов. Она состоит из взаимосвязанных узлов (нейронов), организованных в слои.

Многослойный перцептрон – это базовая модель нейронной сети, которая состоит из нескольких слоев: входного, одного или нескольких скрытых слоев и выходного слоя. Каждый слой состоит из нейронов, которые обрабатывают входные данные, обрабатывает их через функцию активации и передают их дальше по сети.

Основные компоненты нейронной сети:

- **нейрон**: простая функция с входами, весами и нелинейной функцией активации;
- **слои**: группы нейронов, организованные в последовательности;
- **связи**: соединения между нейронами, которые определяют направление передачи сигнала;

– **веса**: числа, присвоенные каждой связи, которые определяют силу взаимодействия между нейронами;

– **активационные функции**: функции, применяемые к нейронам, которые определяют выход нейрона в зависимости от его входного сигнала;

– **функция потери**: функция, которая измеряет ошибку модели на обучающей выборке;

– **оптимизатор**: алгоритм, который используется для минимизации функции потери и обучения модели.

Архитектура многослойного перцептрона:

– входной слой: принимает входные данные. Количество нейронов во входном слое зависит от размерности входных данных;

– скрытые слои: промежуточные вычисления и моделирования сложных нелинейных отношений между входными и выходными данными;

– выходной слой: выдаёт результат работы сети. Количество нейронов в выходном слое зависит от задачи, которую решает сеть.

Функция активации — это нелинейное преобразование, которое применяется к входным данным в каждом слое нейронной сети. Она позволяет моделировать сложные нелинейные зависимости между входными и выходными данными.

– сигмоидальная (sigmoid) функция позволяет получить значения в диапазоне от 0 до 1. Она используется для моделирования вероятности или уверенности;

– функция ReLU (Rectified Linear Unit) возвращает значение входного сигнала, если оно положительное, и 0 в противном случае. Она используется для ускорения обучения и предотвращения проблемы затухания градиента;

– функция Tanh сопоставляет входные значения с диапазоном от -1 до 1. Используется, когда данные сосредоточены вокруг нуля.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями.
2. Выполнить ход работы.
3. Восстановить пример в Google.Colab.
4. Выполнить индивидуальное задание.
5. Ответить на контрольные вопросы.

1 Ход работы

Для начала работы вам необходимо подключить набор данных Diabetes. Ниже представлен код для подключения. Убедитесь, что в наборе данных нет нулевых значений и тип данных признаков числовой. Определите переменные *X* и *Y*. После разделите данные на обучающую и тестовую выборки.

```
data = pd.read_csv('/content/sample_data/diabetes.csv')
X = data.drop('Outcome', axis=1)
y = data['Outcome']
# Разделение данных на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)
```

Затем выполните стандартизацию данных, она необходима для корректного сравнения значений наблюдений, собранных в различных условиях. Она устраняет возможные отклонения и приводит данные к стандартному распределению с нулевым средним и стандартным отклонением, равным 1.

```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Вам нужно создать модель многослойного перцептрона. На вход функция принимает 4 параметра:

- num_hidden_layers: количество скрытых слоев в сети;
- num_neurons_per_layer: количество нейронов в каждом скрытом слое;

– activation: активационная функция, применяемая к нейронам. По умолчанию используется relu (ректифицированная линейная функция);

– optimizer: оптимизатор, используемый для обучения модели. По умолчанию используется adam.

Также создайте объект модели model с помощью keras.Sequential(). Это означает, что будет использоваться последовательная модель, где слои располагаются друг за другом.

Добавьте входной слой, затем скрытые слои, выходной слой. Затем выполните компиляцию модели. Данная функция позволяет экспериментировать с разными архитектурами и параметрами. Ниже представлен код реализации функции:

```
def create_model(num_hidden_layers, num_neurons_per_layer,
activation='relu', optimizer='adam'):
    model = keras.Sequential()
    model.add(layers.Dense(num_neurons_per_layer, activation=activation,
input_shape=(X_train.shape[1],)))
    for _ in range(num_hidden_layers):
        model.add(layers.Dense(num_neurons_per_layer,
activation=activation))
    model.add(layers.Dense(1, activation='sigmoid'))
    model.compile(optimizer=optimizer, loss='binary_crossentropy',
metrics=['accuracy'])
    return model
```

Теперь обучите созданную вами модель, используя код ниже. Для начала задайте количество скрытых слоев, количество нейронов в каждом слое, функцию активации и оптимизатор. Обучите модель на тренировочных данных в течении 100 эпох. Эпоха - один проход через весь обучающий набор для улучшения модели.

```
model = create_model(num_hidden_layers, num_neurons_per_layer, activation,
optimizer)
history = model.fit(X_train, y_train, epochs=100, =0)  # Обучение модели
y_pred = (model.predict(X_test) > 0.5).astype(int)
```

Далее выполните оценку работы модели с помощью следующих метрик: accuracy (точность), precision (точность), recall (полнота) и F1-score (F1-мера).

Использование нескольких метрик позволяет всесторонне оценить производительность модели машинного обучения.

– Accuracy: Доля правильно предсказанных классов;

– Precision: Доля истинно положительных предсказаний среди всех положительных;

– Recall: Доля истинно положительных предсказаний среди всех реальных положительных;

– F1-Score: Гармоническое среднее между точностью и полнотой.

```
accuracy = accuracy_score(y_test, y_pred)
precision = MISS(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
# Вывод результатов
print(f"Точность: {accuracy:.4f}, Точность: {precision:.4f}, Полнота:
{recall:.4f}, F1-мера: {f1:.4f}")
```

Визуализируйте результаты, ниже представлен код для создания графика и вывода эпох.

```
plt.figure(figsize=(10, 6))
plt.plot(history.history['accuracy'])
plt.title(f"Параметры: num_hidden_layers={num_hidden_layers},
num_neurons_per_layer={num_neurons_per_layer}, activation={activation},
optimizer={optimizer}")
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.show()
```

2 Задания на лабораторную работу

Восстановите пропуски (MISS) в блокноте, ориентируясь на информацию из хода работы. Измените модель согласно варианту из таблицы 1, используя первое значение:

1. Настройте кол-во скрытых слоев.
2. Настройте кол-во нейронов в слое.
3. Выберите актив. функцию.
4. Выберите оптимизатор.
5. Оцените эффективность модели.

Повторите для второго значения из таблицы вариантов 1, сравните полученные результаты.

Таблица 1 – Варианты индивидуального задания

Вариант	Кол-во скрытых слоев	Кол-во нейронов в слое	Актив. функция	Оптимизатор
1	1 / 2	16 / 32	relu / tanh	adam / sgd
2	3 / 1	64 / 16	relu / sigmoid	adam / rmsprop
3	2 / 1	32 / 16	relu / tanh	sgd / adam
4	3 / 2	32 / 16	relu / sigmoid	sgd / rmsprop
5	1 / 2	16 / 32	relu / tanh	sgd / adam
6	1 / 2	16 / 64	sigmoid / relu	rmsprop / sgd
7	2 / 3	32 / 16	tanh / relu	adam / sgd
8	1 / 2	16 / 32	relu / sigmoid	sgd / adam
9	2 / 3	32 / 64	sigmoid / tanh	adam / rmsprop
10	1 / 3	64 / 16	relu / tanh	adam / sgd

Контрольные вопросы

1. Что такое нейронная сеть?
2. Что такое многослойный перцептрон (MLP)?
3. Что такое функция потерь?
4. Что такое оптимизатор и для чего он используется?
5. Как влияют различные параметры модели (количество слоев, нейронов, активационные функции, оптимизаторы) на качество обучения и предсказания?
6. Какая комбинация параметров показала лучшие результаты в вашем варианте?
7. Какие бывают адаптивные оптимизаторы нейронной сети и чем они отличаются?
8. Чем отличаются активационные функции?

ЛАБОРАТОРНАЯ РАБОТА № 7

Прогнозирование с помощью LSTM: Анализ и оптимизация

Цель работы: изучить процесс создания и обучения модели LSTM, понять влияние предобработки данных и оптимизации параметров модели на точность прогнозирования.

Краткие теоретические сведения

Рекуррентные нейронные сети (RNN**)** — это особый тип нейронных сетей, где информация передаётся не только между слоями, но и через время. Это позволяет им запоминать и использовать информацию из предыдущих шагов, что особенно полезно при работе с последовательностями данных, такими как текст или временные ряды.

Долгая краткосрочная память (Long Short-Term Memory, LSTM) является разновидностью архитектуры рекуррентных нейронных сетей, разработанной для решения проблемы долговременной зависимости в данных.

Основные компоненты LSTM:

– состояние ячейки (Cell State): это основная память сети, которая хранит информацию на протяжении всего процесса обучения;

– фильтры: включают в себя входной фильтр, фильтр забывания и выходной фильтр. Они контролируют поток информации в и из состояния ячейки, используя сигмоидальные нейронные сети и операцию поточечного умножения.

Принцип работы LSTM:

– забывание: Фильтр забывания определяет, какую информацию из предыдущего состояния ячейки следует забыть;

– обновление: Входной фильтр решает, какую новую информацию следует добавить в состояние ячейки;

– выход: Выходной фильтр определяет, какая часть информации из состояния ячейки должна быть использована для вычисления выходных данных.

Вариации LSTM:

– смотровые глазки (Peephole Connections): позволяют слоям фильтров видеть состояние ячейки;

– объединенные фильтры: объединяют фильтры забывания и входные фильтры для принятия решений о сохранении и обновлении информации;

– управляемые рекуррентные нейроны (GRU): имеют меньше фильтров и немного отличаются соединениями, что делает их более простыми и быстрыми в использовании.

Для обучения модели функция принимает набор данных, число эпох и число нейронов в качестве аргументов.

Метод скользящего окна (sliding window) - это способ преобразования временных рядов в набор данных, пригодный для обучения моделей машинного обучения, которые работают с последовательностями данных.

Принцип работы:

1. Определение размера окна: выбирается размер окна n, который представляет собой количество последовательных точек данных, составляющих одно временное окно.

2. Сдвиг окна: Окно последовательно сдвигается по временным данным.

3. Создание пар "входные данные - выходные данные": для каждого положения окна создается пара:

а) Входные данные: Значения временных рядов в пределах текущего окна;

б) Выходные данные: Целевое значение, которое нужно предсказать. Обычно это значение временного ряда, следующее за последней точкой в окне.

Требования к выполнению лабораторной работы

1. Ознакомиться с методическими указаниями.
2. Выполнить ход работы.
3. Выполнить индивидуальное задание.
4. Ответить на контрольные вопросы.

1 Ход работы

1.1 Загрузка данных

Импортируйте необходимые библиотеки и подключите набор данных для обучения и тестирования модели.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from sklearn.metrics import mean_squared_error, mean_absolute_error,
mean_absolute_percentage_error
import itertools
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
# Загрузка обучающей и тестовой выборок
train_df = pd.read_csv('DailyDelhiClimateTrain.csv', parse_dates=['date'],
index_col=['date'])
test_df = pd.read_csv('DailyDelhiClimateTest.csv', parse_dates=['date'],
index_col=['date'])
```

1.2 Предобработка данных

С помощью метода describe() выведите краткую сводную статистику числовых столбцов набора данных

Также можно выводить определенную информацию для конкретного признака. Попробуйте вывести максимальное, минимальное и среднее арифметическое значение. Эти показатели помогают получить быстрое представление о данных в столбце. Максимальное и минимальное значения показывают, какой диапазон значений присутствует в столбце (рисунок 1.1).

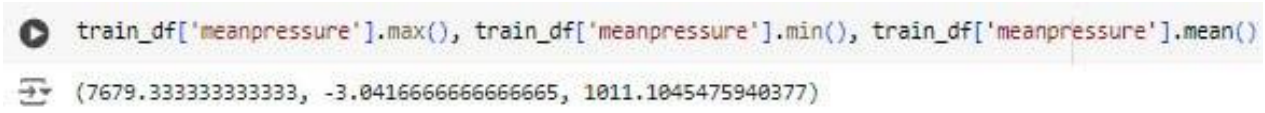

Рисунок 1.1 – Вывод значений

Теперь отфильтруйте тренировочную выборку train_df. Нормальное атмосферное давление на уровне моря составляет около 1013 гПа. Для обучения данных мы будем использовать диапазон значений от 950 до 1050 гПа. Фильтрация данных в этом диапазоне, вероятно, удалит большую часть данных, что позволит избавиться от выбросов. Это, в свою очередь может сделать модель более точной.

```
train_df = train_df[(train_df['meanpressure'] > 950) & (train_df['meanpressure'] < 1090)]
```

Постройте графики для анализа трендов, сезонности, цикличности, аномалий и других важных особенностей.

После анализа набора данных можно приступать к обучению модели. Поделите данные на тренировочную и тестовую части. Выполните стандартизацию (масштабирование меток), для улучшения сходимости нейронной сети. LSTM обычно работает лучше и быстрее сходится, если входные данные находятся в диапазоне от 0 до 1. Это связано с тем, что LSTM использует градиентный спуск для обновления весов. Если целевые значения сильно отличаются по величине от признаков, это может привести к неустойчивому обучению или замедлению сходимости. Временные ряды имеют непрерывную зависимость между шагами, и для лучшего понимания этой зависимости рекомендуется согласовать масштаб данных. Это улучшает понимание временных закономерностей и повышает точность прогнозов.

```
train_size = int(len(data) * 0.75)
test_size = len(data) - train_size
print("Train Size :",train_size,"Test Size :",test_size)
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler(feature_range= (0, 1))
scaled_data = scaler.fit_transform(np.array(data))
train_data = scaled_data[0:train_size, :]
```

```
train_data.shape
```

Теперь необходимо определить размер временного окна и закончить подготовку данных для LSTM. Длина окна должна быть достаточно большой, чтобы охватить временные паттерны, важные для предсказания. Если данные о погоде демонстрируют месячные циклы или колебания, окно в 60 дней помогает уловить эти повторяющиеся тренды. Окно можно корректировать, подбирая оптимальное значение для конкретной задачи.

```
time_steps = 60
test_data = scaled_data[train_size - time_steps:, :]
x_test = []
y_test = []
n_cols = 1
for i in range(time_steps, len(test_data)):
    x_test.append(test_data[i-time_steps:i, 0:n_cols])
    y_test.append(test_data[i, 0:n_cols])
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], n_cols))
```

1.3 Работа с моделью

После подготовки набора данных нужно создать модель рекуррентной нейронной сети типа LSTM для предсказания значений *meantemp*.

В блокноте пропущено создание модели LSTM, вам необходимо её восстановить.

Теперь выполните последние настройки обучающих данных модели. Представленный код снизу разделяет временной ряд на последовательности и обучает RNN, чтобы предсказывать следующее значение на основе предыдущих.

```
x_train = []
y_train = []
time_steps = 60
n_cols = 1
for i in range(time_steps, len(scaled_data)):
    x_train.append(scaled_data[i-time_steps:i, :n_cols])
    y_train.append(scaled_data[i, :n_cols])
    if i<=time_steps:
        print('X_train: ', x_train)
```

```
        print('y_train:' , y_train)
# Преобразовать в массив numpy
x_train, y_train = np.array(x_train), np.array(y_train)
# Изменение входных данных на (n_samples, time_steps, n_feature)
# Подготовка данных для использования в модели LSTM
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], n_cols))
# Обучение модели
history = model2.fit(x_train, y_train, epochs= 100, batch_size= 32)
```

Подготовьте тестовые данные для проверки модели. Для этого выберите размер окна, количество признаков (*n_cols*). Для *test_data* выбирается часть нормированных данных с точки, которая находится на расстоянии 60 шагов от начала тестового набора. В цикле каждая последовательность в *x_test* используется для предсказания следующего значения в *y_test*.

```
time_steps = 60
test_data = scaled_data[train_size - time_steps:, :]
x_test = []
y_test = []
n_cols = 1
for i in range(time_steps, len(test_data)):
    x_test.append(test_data[i-time_steps:i, 0:n_cols])
    y_test.append(test_data[i, 0:n_cols])
x_test, y_test = np.array(x_test), np.array(y_test)
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], n_cols))
```

1.4 Результат модели

Постройте графики для визуализации результатов обучения модели. На рисунке 1.2 представлен результат обучения модели.

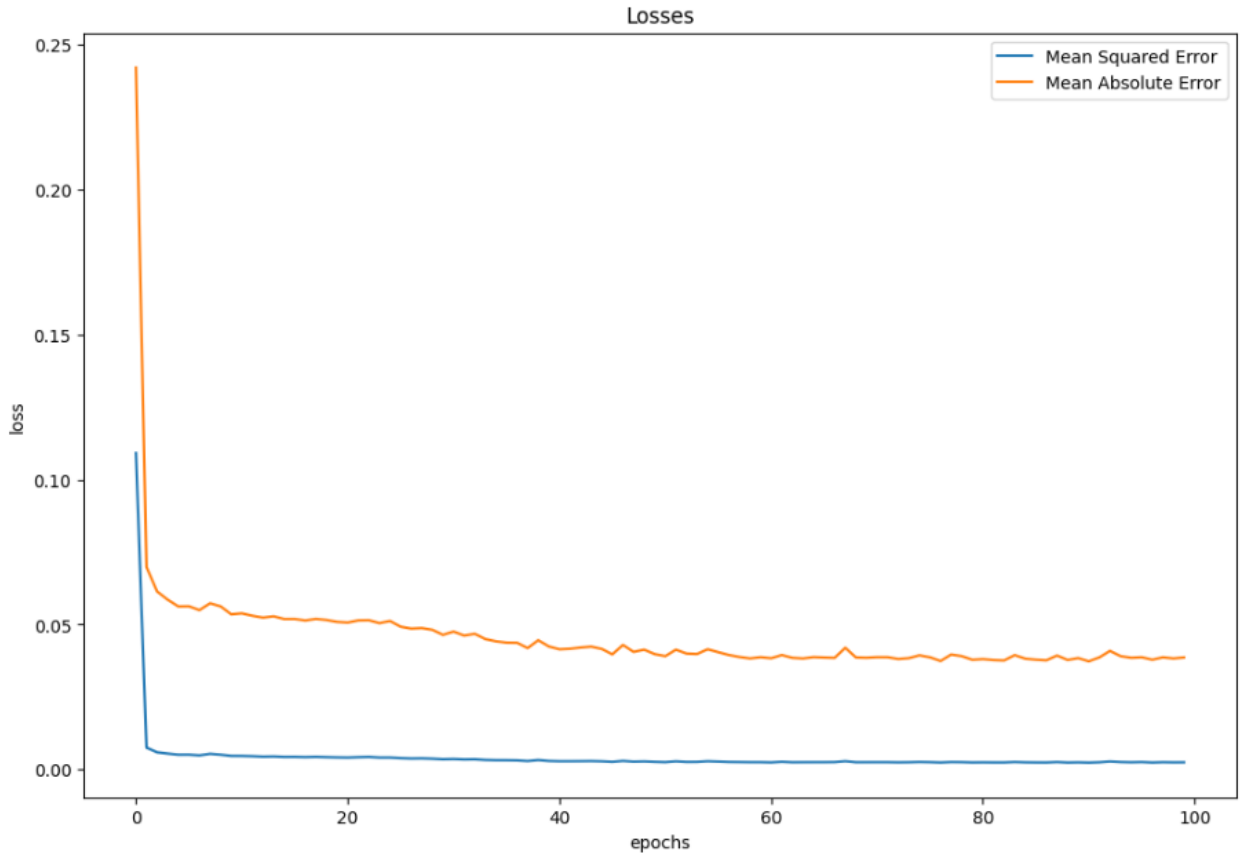

Рисунок 1.2 – График потерь

Восстановите исходный масштаб прогнозов и истинных значений, чтобы правильно интерпретировать результаты.

```
predictions = model2.predict(x_test)
predictions = scaler.inverse_transform(predictions)
predictions.shape
y_test = scaler.inverse_transform(y_test)
```

Теперь используйте метрику RMSE для оценки точности модели. RMSE — метрика, которая измеряет среднее расстояние между предсказанными и истинными значениями. Чем ниже RMSE, тем лучше модель. Результат у полученной модели равняется 0.31.

```
RMSE = np.sqrt(np.mean( y_test - predictions )**2).round(2)
```

На рисунках 1.3 – 1.4 представлены графики результата работы модели.

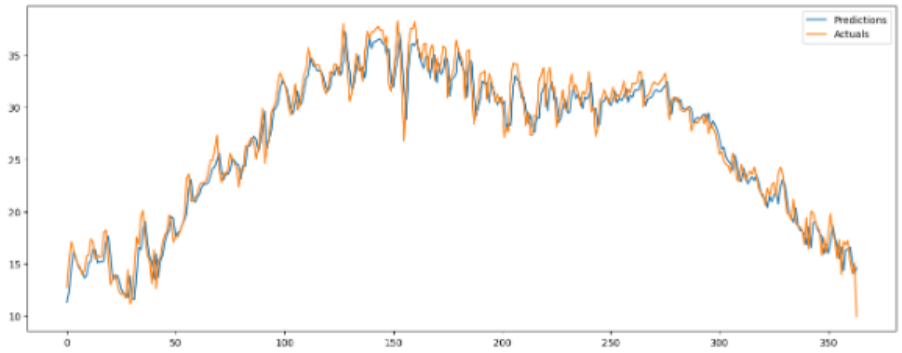

Рисунок 1.3 – График предсказаний

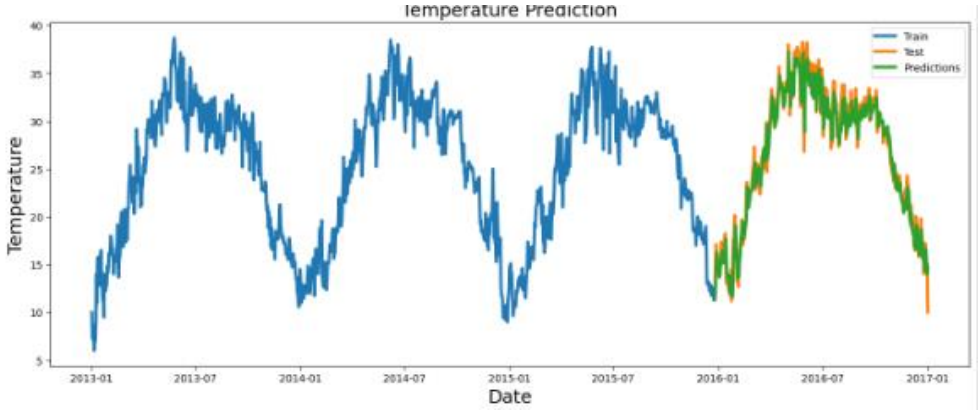

Рисунок 1.4 – График предсказания температуры

Выполните обучение и тестирование модели без масштабирования данных и сравните результаты.

2 Задания на лабораторную работу

В ходе выполнения индивидуального задания, вам необходимо выполнить следующие шаги:

1. Подготовьте набор данных для настройки и тестирования модели. Набор необходимо выбрать из таблицы 1, согласно варианту.
2. Реализуйте модель рекуррентной нейронной сети типа LSTM для вашего набора данных.
3. Обучите модель на обучающем наборе данных.
4. Протестируйте настроенную модель.

5. Точность модели должна быть не менее 50%.
6. Постройте графики.

Таблица 1 – Варианты индивидуального задания

Вариант	Набор данных
1	https://www.kaggle.com/datasets/uciml/electric-power-consumption-data-set/data/download
2	https://www.kaggle.com/datasets/mnassrib/jena-climate/data/download
3	https://www.kaggle.com/datasets/shreenidhihipparagi/google-stock-prediction/data/download
4	https://www.kaggle.com/datasets/rupakroy/lstm-datasets-multivariate-univariate/data/download
5	https://www.kaggle.com/datasets/jacksoncrow/stock-market-dataset/data/download
6	https://www.kaggle.com/datasets/rohith203/traffic-volume-dataset/data/download
7	https://www.kaggle.com/datasets/robervalt/sunspots/data/download
8	https://www.kaggle.com/datasets/sudalairajkumar/daily-temperature-of-major-cities/data/download

Контрольные вопросы

1. Какая функция активации использовалась в LSTM-слоях?
2. Какие компоненты есть в ячейке LSTM?
3. В чем основное преимущество LSTM перед стандартной RNN?
4. Объясните, что такое рекуррентные нейронные сети (RNN) и как они отличаются от обычных нейронных сетей.
5. Объясните принцип работы метода скользящего окна для создания временных рядов.
6. Что такое RMSE?
7. Какие методы оптимизации моделей машинного обучения вам известны?

Литература

1. Замятин, Н. В. Нечеткая логика и нейронные сети: Учебное пособие / Замятин Н. В. — Томск: ТУСУР, 2014. — 289 с.

2. Замятин, Н. В. Системы искусственного интеллекта: Учебное пособие / Н. В. Замятин. — Томск: ТУСУР, 2018. — 244 с.

3. Филипова, И. А. Правовое регулирование искусственного интеллекта : учебное пособие / И. А. Филипова. — Нижний Новгород : ННГУ им. Н. И. Лобачевского, 2020. — 90 с.

4. Джонс, М. Т. Программирование искусственного интеллекта в приложениях / М. Т. Джонс. — Москва : ДМК Пресс, 2011. — 312 с