Федеральное государственное автономное

образовательное учреждение высшего образования

«Пермский государственный национальный исследовательский университет» (ПГНИУ)

Региональный институт непрерывного образования (РИНО ПГНИУ)

Цифровая кафедра

Выпускная аттестационная (квалификационная) работа

по курсу профессиональной переподготовки

«Анализ данных и машинное обучение»

**КЛАССИФИКАЦИЯ ПРОБЛЕМ ЭЛЕКТРИЧЕСКОЙ СЕТИ.**

Разработчики проекта:

(ФИО)

Лягаев Александр Денисович

Пермь, 2024

**ПАСПОРТ ПРОЕКТА**

**Название проекта:** Классификация проблем электрической сети

**Сведения об авторах:** Лягаев Александр Денисович

**Цель:** Выполнить анализ данных и построить модель, на основе которой можно предсказать класс проблемы в электрической сети.

**Задачи:**

* Выполнить анализ проблемы, обосновать ее актуальность.
* Осуществить загрузку данных и подготовку их к анализу.
* Осуществить моделирование зависимости целевого признака от остальных с помощью различных классификаторов, подобрать наилучшую модель, оценить ее качество и выполнить прогнозирование.
* Выбрать наилучшую модель, сделать выводы.

**Краткое описание проекта:**

Проект направлен на построение и обучение модели, предназначенной для классификации проблем в электросети. Основная цель – создание инструмента, который поможет идентифицировать тип неисправности.

**Конкретные ожидаемые результаты:**

Построенная модель классификации.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

Линия электропередачи является наиболее важной частью энергосистемы. Потребность в мощности и ее преданность возросли в геометрической прогрессии в современную эпоху, и важнейшая роль линии электропередачи заключается в передаче электроэнергии из области источника в распределительную сеть. Электроэнергетическая система состоит из множества сложных динамических и взаимодействующих элементов, которые всегда подвержены помехам или электрическим неисправностям.

Мощные электростанции и концепция сети, т. е. синхронизированные электростанции и географически смещенные сети, требовали обнаружения неисправностей и срабатывания защитного оборудования, чтобы оставаться стабильными.

Неисправности на линиях электропередачи должны быть сначала обнаружены и правильно классифицированы, а после быть устранены за минимально возможное время.

Система защиты, используемая для линии электропередачи, также может инициировать другие реле для защиты энергосистемы от отключений.

Образцовая система обнаружения неисправностей обеспечит практичный, надежный, быстрый и безопасный способ работы релейной защиты. Применение технологии распознавания образов, может помочь отличить неисправные и исправные электроэнергетические системы.

Для решения этой задачи используем искусственные нейронные сети (ИНС), так как они эффективны в определении неисправного шаблона и классификации неисправностей путем распознавания образов.

**ИСХОДНЫЕ ДАННЫЕ**

В дальнейшей работе используется набор данных о разрывах в электрической сети.

Набор данных состоит из 10 столбцов и содержит информацию о линиях разрыва, о линейном токе и линейном напряжении фаз:

* G - Часть линии разрыва (Земля) если 1, иначе 0
* C - Часть линии разрыва (Фаза C) если 1, иначе 0
* B - Часть линии разрыва (Фаза B) если 1, иначе 0
* A - Часть линии разрыва (Фаза A) если 1, иначе 0
* Ia - Линейный ток фазы А
* Ib - Линейный ток фазы B
* Ic - Линейный ток фазы C
* Va - Линейное напряжение фазы А
* Vb - Линейное напряжение фазы B
* Vc - Линейное напряжение фазы C

Классификация разрывов составляется из столбцов G, C, B, A. Всего классов 6, примеры:

* [0 0 0 0] - Нет проблем
* [1 0 0 1] - LG (разрыв между фазой A и G (землёй))
* [0 0 1 1] - LL (разрыв между фазой A и фазой B)
* [1 0 1 1] - LLG (разрыв между фазами A,B и G (землёй))
* [0 1 1 1] - LLL (разрыв между всеми тремя фазами)
* [1 1 1 1] - LLLG (разрыв между всеми тремя фазами и G (землёй))

Необходимо проанализировать данные о разрывах и определить, возможно ли описать зависимость классов от имеющихся характеристик различными методами.

**РЕАЛИЗАЦИЯ ПРОЕКТА**

**Этап 1. Подготовка данных к анализу**

Подключаем необходимые библиотеки:

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_palette('husl')

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

from imblearn.pipeline import Pipeline

from imblearn.under\_sampling import RandomUnderSampler

Загрузим данные в объект DataFrame:

url='https://raw.githubusercontent.com/KoJl9lH-59rus/ML\_FINAL/refs/heads/main/classData.csv'

dataset = pd.read\_csv(url)

dataset.head()

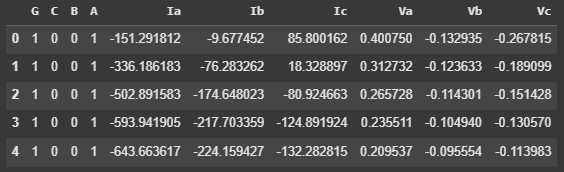


Рисунок 1. Исходный датафрейм

Убедимся, что все столбцы имеют числовой тип и пустых значений нет.

print(dataset.info(), '\n')

print('Количество пропусков = ', dataset.isnull().sum().sum())

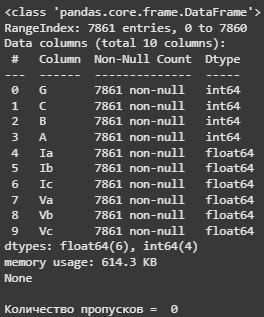


Рисунок 2. Типы данных колонок, количество пропусков

Вспоминаем из чего состоят классы, а именно из столбцов G C B A. Добавим столбец, в который на основании значений четырех столбцов будут присвоены классы, признак станет целевым.

Классы будем рассматривать следующим образом:

0 - Нет неисправности

1 - LL

2 - LLL

3 - LG

4 - LLG

5 - LLLG

fault\_mapping = {

(0,0,0,0): 0, # Нет неисправности

(0,0,1,1): 1, # Неисправность LL

(0,1,0,1): 1, # Неисправность LL

(0,1,1,0): 1, # Неисправность LL

(0,1,1,1): 2, # Неисправность LLL

(1,0,0,1): 3, # Неисправность LG

(1,0,1,0): 3, # Неисправность LG

(1,0,1,1): 4, # Неисправность LLG

(1,1,0,0): 3, # Неисправность LG

(1,1,0,1): 4, # Неисправность LLG

(1,1,1,0): 4, # Неисправность LLG

(1,1,1,1): 5, # Неисправность LLLG

}

# Применяем замену

dataset['Class'] = dataset[['G', 'C', 'B', 'A']].apply(lambda x: fault\_mapping[tuple(x)], axis=1)

dataset = dataset.drop(columns=['G', 'C', 'B', 'A'])

dataset.head()

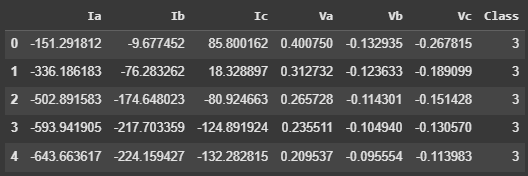


Рисунок 3. Результат замены столбцов одним

Выведем основные статистические показатели

dataset.describe()

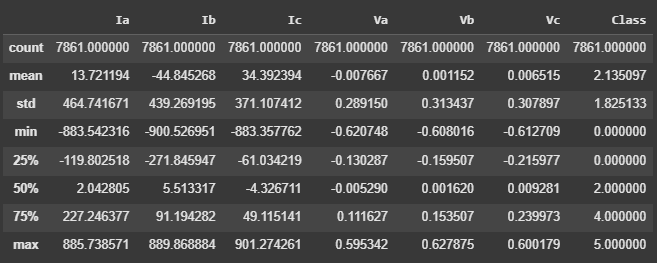


Рисунок. 4. Основные статистические показатели

На предоставленном рисунке можно увидеть следующие параметры для числовой переменной: count – количество наблюдений, mean – среднее значение, std – стандартное отклонение. Стандартное отклонение характеризует средней отклонение от среднего значения. 25%, 50%, 75% - нижний, серединный(медиана) и верхний квартиль, а max – максимальное значения.

Проведём нормализацию данных, потому что у нас параметры тока и напряжения разных порядков. Если этого не сделать, то может произойти так, что модель отдаст предпочтение признакам, у которых большие значения. Это может сказаться на предсказательной способности модели и повлиять на метрики.

Перед тем как провести нормализацию данных необходимо убрать целевую переменную из датафрейма и нормализовать остальные:

from sklearn.preprocessing import MinMaxScaler

columns\_to\_normalize = ['Ia', 'Ib', 'Ic', 'Va', 'Vc']

# Создаем экземпляр MinMaxScaler

scaler = MinMaxScaler()

# Нормализуем только выбранные столбцы

dataset[columns\_to\_normalize] = scaler.fit\_transform(dataset[columns\_to\_normalize])

# Выводим результат

dataset.head()

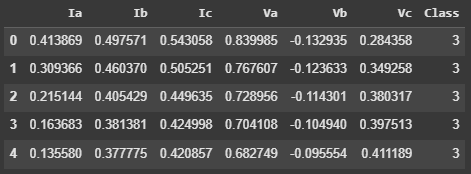


Рисунок 5. Результат нормализации

**Этап 2. Предварительный анализ данных**

Для классификации очень важным является наличие или отсутствие баланса в количестве различных классов. Построим диаграмму распределения классов в нашем наборе данных.

ax =plt.subplot(1,1,1)

ax=dataset['Class'].value\_counts().plot.pie(explode=[0.1, 0.1,0.1,0.1, 0.1,0.1],autopct='%1.2f%%',shadow=True);

plt.tight\_layout()

plt.axis('off');

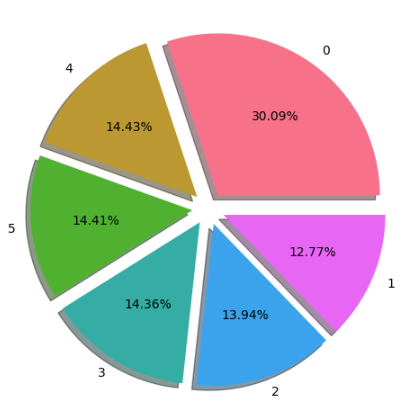


Рисунок 6. Диаграмма распределения классов

Мы видим, что класс несбалансированные – класс 0 количественно превосходит каждый раза в 2.

**Этап 3. Корреляционный анализ данных**

**Корреляционный анализ** – это совокупность методов оценивания степени тесноты статистической связи между анализируемыми переменными.

Построим тепловую карту для визуализации матрицы корреляции по методу Пирсона для нашего набора данных, с помощью которой можно посмотреть корреляцию между переменными, то есть отследить зависимость между двумя переменными

plt.figure(figsize=(7,6))

sns.heatmap(dataset.corr(), annot = True, cmap = 'coolwarm')

plt.show()

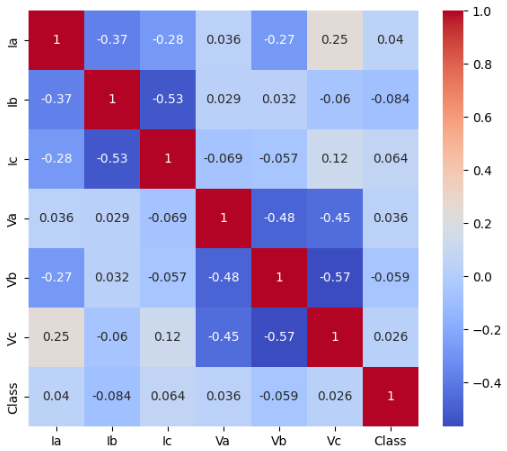


Рисунок 7. Матрица корреляции

Можно заметить, что целевая переменная имеет довольно слабую корреляцию с другими.

**Этап 4. Создание и обучение моделей классификаторов.**

Сначала мы определяемся с теми классификаторами, которые мы хотим обучить, чтобы выбрать из них наилучший вариант. Были рассмотрены следующие варианты:

* Логистическая регрессия (LogisticRegression)
* Метод К-ближайших соседей (KNeighborsClassifier)
* Гауссовский наивный байесовский классификатор (Gaussian Naive Bayes)
* Дерево решений (DecisionTreeClassifier)
* Градиентный бустинг (Gradient Boosting Classifier)
* Метод опорных векторов (SVC)
* Метод случайного леса (Random Forest Classifier)

tuned\_classifiers = {

"Logistic Regression": LogisticRegression(max\_iter=10000, random\_state=42),

"KNearest": KNeighborsClassifier(n\_neighbors=6),

"Gaussian Naive Bayes": GaussianNB(),

"Decision Tree Classifier": DecisionTreeClassifier(random\_state=42),

"Gradient Boosting Classifier": GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.4, max\_depth=5, random\_state=42),

"Support Vector Classifier": SVC(kernel='rbf', gamma='auto', C=1.0),

"Random Forest Classifier": RandomForestClassifier(n\_estimators=100, max\_depth=None, random\_state=42),

}

Разбиваем данные на тренировочную и тестовые выборки, тренировочная составляет 80%, а тестовая 20%

Устраняем дисбаланс в распределении классов с помощью метода **RandomUnderSampler**. Этот метод случайным образом выбирает экземпляры из класса с более высоким количеством, чтобы уменьшить его размер до уровня минимального класса. Это означает, что выбираются случайные подмножества данных, чтобы сбалансировать классы.

X = dataset.drop(columns=['Class'])

y = dataset['Class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 42)

# Используем StratifiedKFold для обеспечения стратифицированной кросс-валидации

# Это объект кросс-валидации, который позволяет сохранить процент образцов для каждого класса в каждом фоне.

skf = StratifiedKFold(n\_splits=5, random\_state=None, shuffle=True)

# 5 образцов для кросс-валидации

# Создаем объект для уменьшения выборки для большинства

sampler = RandomUnderSampler()

**Кросс-валидация** — это метод оценки качества модели, который позволяет проверить ее способность обобщать, избегая переобучения.

**Этап 5. Подбор модели**

**Pipeline** — инструмент для организации последовательной обработки данных, упрощающий написание, отладку и сопровождение кода, особенно в контексте машинного обучения.

import time

accuracy\_data = []

for key, classifier in tuned\_classifiers.items():

time1 = time.time()

# Создаем список шагов для пайплайна машинного обучения

steps = [ ('u', sampler), ('model', classifier)] # Шаги: увеличенная выборка, уменьшенная выборка и модель

# Создаем пайплайн машинного обучения

FiPipeline = Pipeline(steps=steps)

# Кросс-валидация классификаторов с использованием определенного пайплайна

scores = cross\_val\_score(FiPipeline, X\_train, y\_train, cv=skf)

time2 = time.time()

accuracy = round(scores.mean() \* 100, 2)

accuracy\_data.append([key, accuracy])

print(f"Название: {key}, оценка точности: {accuracy} %, время обучения : {(time2-time1):0.2f} секунд")

Также попробуем обучить рекуррентную нейросеть. Классификацию на основе нейронных сетей создаем самостоятельно при помощи библиотеки **tensorflow**

y\_train\_cat = to\_categorical(y\_train, num\_classes=6)

y\_test\_cat = to\_categorical(y\_test, num\_classes=6)

# Создание модели

model\_n = Sequential()

model\_n.add(Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)))

model\_n.add(Dense(32, activation='relu'))

model\_n.add(Dense(16, activation='relu'))

model\_n.add(Dense(6, activation='sigmoid'))

# Компиляция модели, задача многоклассовая классификация

model\_n.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Обучение модели

history = model\_n.fit(X\_train, y\_train\_cat,

epochs=50,

validation\_data=(X\_test, y\_test\_cat))

# Оценка модели

test\_loss, test\_acc = model\_n.evaluate(X\_test, y\_test\_cat)

print()

accuracy = round(test\_acc, 4)\*100

accuracy\_data.append(["Нейросеть",accuracy])

print(f'Тестовая точность: {accuracy}')

print()

Визуализируем данные обучения.

import matplotlib.pyplot as plt

# Получаем название + точность

models = [item[0] for item in accuracy\_data]

accuracies = [item[1] for item in accuracy\_data]

colors = plt.cm.viridis(np.linspace(0, 1, len(models)))

plt.figure(figsize=(9, 5))

bar\_width = 1

bars = plt.barh(models, accuracies, height=bar\_width, color=colors)

plt.xlabel('Accuracy (%)')

plt.title('Model Accuracy Comparison')

plt.yticks(models)

plt.xlim(0, 100)

def autolabel(bars):

for bar in bars:

width = bar.get\_width()

plt.annotate(f'{width:.2f}%',

xy=(width, bar.get\_y() + bar.get\_height() / 2),

xytext=(3, 0),

textcoords="offset points",

ha='left', va='center')

autolabel(bars)

plt.tight\_layout()

plt.show()

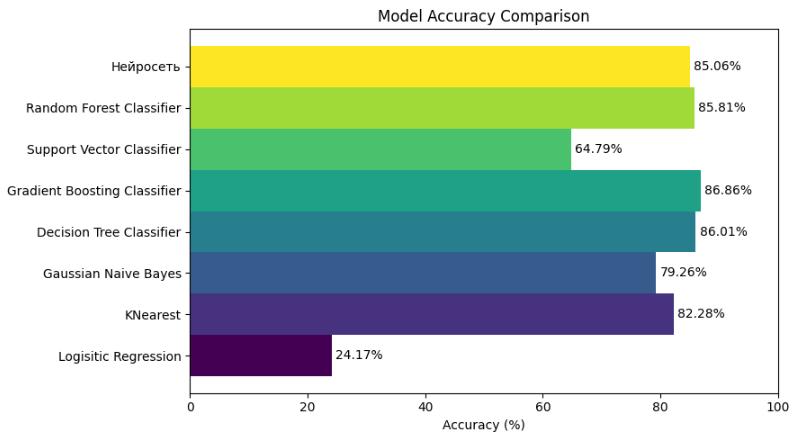


Рисунок 8. Сравнение точности моделей

По итогам обучения можно увидеть, что ни одна из моделей не обучилась идеально. Результаты не превосходят 87%, наиболее точным классификатором оказался Градиентный бустинг, потому обучим эту модель и выведем результаты обучения в виде отчета о классификации и матрицы ошибок.

**Этап 6. Обучение модели**

Обучим модель и выведем матрицу ошибок

model = Pipeline([

('u', RandomUnderSampler()),

("Gradient Boosting", GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.4, max\_depth=5, random\_state=42)) ])

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

prediction = model.predict(X\_test)

# Вывод метрик

labels = ['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)']

print(classification\_report(y\_test, prediction, target\_names=labels))

conf\_matrix = confusion\_matrix(y\_test, prediction)

cm\_df = pd.DataFrame(conf\_matrix, index=['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)'], columns=['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)'])

# Настройка отображения без научной нотации

pd.options.display.float\_format = '{:.0f}'.format

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_df, annot=True, fmt='g', cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Plot")

plt.show()

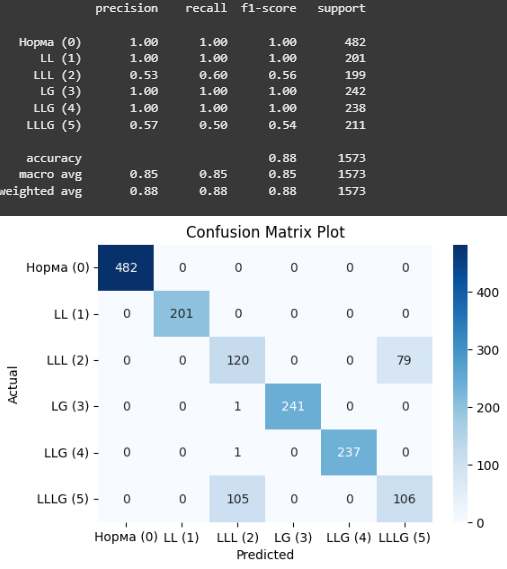


Рисунок 9. Результат обучения модели

Анализируя полученные результаты можно сделать вывод, что состояние электросети может быть предсказано с помощью модели машинного обучения.

Внедрение машинного обучения в процесс анализа состояния электросети значительно упрощает и ускоряет процесс. Модели, основанные на анализе данных, могут учитывать необходимые факторы, включая линейный ток и линейное напряжение, что повышает уровень устойчивости системы.

Большинство моделей машинного обучения продемонстрировали схожий уровень точности, однако наивысшие результаты были показаны моделью градиентного бустинга. Таким образом, поставленная цель была достигнута: на основе разрывов в электрической сети разработаны модели классификации. Но для такой важной отрасли как электроэнергетика, модель показывает не очень хорошие результаты относительно выявления проблем LLL и LLLG, у моделей плохо получается отличить данные разрывы друг от друга.

Построенная модель может быть применена для анализа электросетей, но с учетом имеющихся проблем.

**СПИСОК ЛИТЕРАТУРЫ**

* Сузи Р.А. Язык программирования Python: учебное пособие / Р.А. Сузи. – 3-е изд. – М.: Интернет-Университет Информационных Технологий (ИНТУИТ), Ай Пи Ар Медиа, 2020. – 350 c.
* Документация по библиотеке scikit-learn <https://scikit-learn.org/stable/api/sklearn.html>
* Учебные пособия по машинному обучению <https://colab.research.google.com/drive/1YIOGf38skLQ-RX5TFojpyXdaBy9NlvlF?usp=sharing>
* Документация по библиотеке TensorFlow <https://www.tensorflow.org/api_docs/python/tf>
* Electrical Fault detection and classification <https://www.kaggle.com/datasets/esathyaprakash/electrical-fault-detection-and-classification/data>

**Приложение**

Программный код

import numpy as np

import pandas as pd

import seaborn as sns

sns.set\_palette('husl')

import matplotlib.pyplot as plt

%matplotlib inline

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import accuracy\_score

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import classification\_report, confusion\_matrix

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from tensorflow.keras.utils import to\_categorical

from imblearn.pipeline import Pipeline

from imblearn.under\_sampling import RandomUnderSampler

#Загружаем датасет из github

url = 'https://raw.githubusercontent.com/KoJl9lH-59rus/ML\_FINAL/refs/heads/main/classData.csv'

dataset = pd.read\_csv(url)

dataset.shape

dataset.head()

print(dataset.info(), '\n')

print('Количество пропусков = ', dataset.isnull().sum().sum())

dataset.describe()

fault\_mapping = {

(0,0,0,0): 0, # Нет неисправности

(0,0,1,1): 1, # Неисправность LL

(0,1,0,1): 1, # Неисправность LL

(0,1,1,0): 1, # Неисправность LL

(0,1,1,1): 2, # Неисправность LLL

(1,0,0,1): 3, # Неисправность LG

(1,0,1,0): 3, # Неисправность LG

(1,0,1,1): 4, # Неисправность LLG

(1,1,0,0): 3, # Неисправность LG

(1,1,0,1): 4, # Неисправность LLG

(1,1,1,0): 4, # Неисправность LLG

(1,1,1,1): 5, # Неисправность LLLG

}

# Применяем замену

dataset['Class'] = dataset[['G', 'C', 'B', 'A']].apply(lambda x: fault\_mapping[tuple(x)], axis=1)

dataset = dataset.drop(columns=['G', 'C', 'B', 'A'])

dataset.head()

# Создание countplot

sns.countplot(x=dataset['Class'], color='#4A90E2')

# Задаем метки для классов

class\_labels = ['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)']

plt.xticks(ticks=range(len(class\_labels)), labels=class\_labels)

# Показать график

plt.xlabel('Классы')

plt.ylabel('Количество')

plt.title('Распределение классов')

plt.show()

ax =plt.subplot(1,1,1)

ax=dataset['Class'].value\_counts().plot.pie(explode=[0.1, 0.1,0.1,0.1, 0.1,0.1],autopct='%1.2f%%',shadow=True);

plt.tight\_layout()

plt.axis('off');

plt.figure(figsize=(7,6))

sns.heatmap(dataset.corr(), annot = True, cmap = 'coolwarm')

plt.show()

tuned\_classifiers = {

"Logistic Regression": LogisticRegression(max\_iter=10000, random\_state=42),

"KNearest": KNeighborsClassifier(n\_neighbors=6),

"Gaussian Naive Bayes": GaussianNB(),

"Decision Tree Classifier": DecisionTreeClassifier(random\_state=42),

"Gradient Boosting Classifier": GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.4, max\_depth=5, random\_state=42),

"Support Vector Classifier": SVC(kernel='rbf', gamma='auto', C=1.0),

"Random Forest Classifier": RandomForestClassifier(n\_estimators=100, max\_depth=None, random\_state=42),

}

X = dataset.drop(columns=['Class'])

y = dataset['Class']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state = 42)

# Используем StratifiedKFold для обеспечения стратифицированной кросс-валидации

# Это объект кросс-валидации, который позволяет сохранить процент образцов для каждого класса в каждом фоне.

skf = StratifiedKFold(n\_splits=5, random\_state=None, shuffle=True) # 5 образцов для кросс-валидации

# Создаем объект для уменьшения выборки для большинства

sampler = RandomUnderSampler()

import time

accuracy\_data = []

# Проходим по словарю с классификаторами

for key, classifier in tuned\_classifiers.items():

time1 = time.time()

# Создаем список шагов для пайплайна машинного обучения

steps = [ ('u', sampler), ('model', classifier)] # Шаги: увеличенная выборка, уменьшенная выборка и модель

# Создаем пайплайн машинного обучения

FiPipeline = Pipeline(steps=steps) # Инициализация пайплайна с указанными шагами

# Кросс-валидация классификаторов с использованием определенного пайплайна

scores = cross\_val\_score(FiPipeline, X\_train, y\_train, cv=skf) # Оценка модели с помощью кросс-валидации

time2 = time.time()

accuracy = round(scores.mean() \* 100, 2)

accuracy\_data.append([key, accuracy])

print(f"Название: {key}, оценка точности: {accuracy} %, время обучения : {(time2-time1):0.2f} секунд")

y\_train\_cat = to\_categorical(y\_train, num\_classes=6)

y\_test\_cat = to\_categorical(y\_test, num\_classes=6)

# Создание модели

model\_n = Sequential()

model\_n.add(Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)))

model\_n.add(Dense(32, activation='relu'))

model\_n.add(Dense(16, activation='relu'))

model\_n.add(Dense(6, activation='sigmoid'))

# Компиляция модели, задача многоклассовая классификация

model\_n.compile(optimizer='adam',

loss='categorical\_crossentropy',

metrics=['accuracy'])

# Обучение модели

history = model\_n.fit(X\_train, y\_train\_cat,

epochs=50,

validation\_data=(X\_test, y\_test\_cat))

# Оценка модели

test\_loss, test\_acc = model\_n.evaluate(X\_test, y\_test\_cat)

print()

accuracy = round(test\_acc, 4)\*100

accuracy\_data.append(["Нейросеть",accuracy])

print(f'Тестовая точность: {accuracy}')

print()

import matplotlib.pyplot as plt

# Получаем название + точность

models = [item[0] for item in accuracy\_data]

accuracies = [item[1] for item in accuracy\_data]

colors = plt.cm.viridis(np.linspace(0, 1, len(models)))

plt.figure(figsize=(9, 5))

bar\_width = 1

bars = plt.barh(models, accuracies, height=bar\_width, color=colors)

plt.xlabel('Accuracy (%)')

plt.title('Model Accuracy Comparison')

plt.yticks(models)

plt.xlim(0, 100)

def autolabel(bars):

for bar in bars:

width = bar.get\_width()

plt.annotate(f'{width:.2f}%',

xy=(width, bar.get\_y() + bar.get\_height() / 2),

xytext=(3, 0),

textcoords="offset points",

ha='left', va='center')

autolabel(bars)

plt.tight\_layout()

plt.show()

model = Pipeline([

('u', RandomUnderSampler()),

("Gradient Boosting", GradientBoostingClassifier(n\_estimators=100, learning\_rate=0.4, max\_depth=5, random\_state=42)) ])

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

prediction = model.predict(X\_test)

# Вывод метрик

labels = ['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)']

print(classification\_report(y\_test, prediction, target\_names=labels))

conf\_matrix = confusion\_matrix(y\_test, prediction)

cm\_df = pd.DataFrame(conf\_matrix, index=['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)'], columns=['Норма (0)', 'LL (1)', 'LLL (2)', 'LG (3)', 'LLG (4)', 'LLLG (5)'])

# Настройка отображения без научной нотации

pd.options.display.float\_format = '{:.0f}'.format

plt.figure(figsize=(6, 4))

sns.heatmap(cm\_df, annot=True, fmt='g', cmap="Blues")

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix Plot")

plt.show()