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»

**ПРЕДИКТИВНАЯ СИСТЕМА ВОЗНИКНОВЕНИЯ ОЖИРЕНИЯ.**

Разработчики проекта:

(ФИО)

Соколов Денис Александрович

Пермь, 2024

**ПАСПОРТ ПРОЕКТА**

**Название проекта:** Предиктивная система возникновения ожирения

**Сведения об авторах:** Соколов Денис Александрович

**Цель:** Выполнить анализ данных и построить модель, на основе которой можно предсказать возникновение ожирения.

**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 предсказания возникновения ожирения. Основная цель – создание инструмента, который поможет предсказать тип ожирения.

**Конкретные ожидаемые результаты:**

Построенная модель классификации.

**СОДЕРЖАНИЕ ПРОЕКТА**

**Анализ проблемы исследования**

Ожирение является одной из наиболее актуальных проблем современного здравоохранения, оказывающей значительное влияние на качество жизни и здоровье населения. Распространенность ожирения растет в геометрической прогрессии, что связано с изменением образа жизни, питания и снижением физической активности. Ожирение является не только самостоятельным заболеванием, но и фактором риска развития множества хронических заболеваний, таких как сахарный диабет, сердечно-сосудистые заболевания и другие.

Современные подходы к профилактике и лечению ожирения требуют не только своевременного выявления, но и прогнозирования риска его возникновения. Для этого необходимо учитывать множество факторов, включая генетическую предрасположенность, особенности метаболизма, пищевые привычки, уровень физической активности и психологические аспекты.

Для решения задачи прогнозирования ожирения необходимо разработать систему, которая сможет анализировать сложные взаимосвязи между различными факторами и выявлять закономерности, ведущие к развитию заболевания. Такая система должна быть практичной, надежной и точной, чтобы обеспечивать своевременное вмешательство и профилактику.

Применение технологий искусственного интеллекта, в частности искусственных нейронных сетей (ИНС), может стать эффективным инструментом для решения этой задачи. ИНС способны анализировать большие объемы данных, выявлять сложные паттерны и классифицировать риски на основе распознавания образов. Это позволяет не только прогнозировать вероятность возникновения ожирения, но и разрабатывать персонализированные рекомендации для профилактики и лечения.

Таким образом, разработка предиктивной системы на основе ИНС для прогнозирования ожирения является актуальной задачей, которая может внести значительный вклад в улучшение здоровья населения и снижение нагрузки на систему здравоохранения.

**ИСХОДНЫЕ ДАННЫЕ**

В дальнейшей работе используется набор данных о наличии у человека ожирения и его степени.

Набор данных состоит из 16 столбцов и содержит информацию о различных характеристиках людей, связанных с их весом, образом жизни и привычками. Вот описания каждого столбца:

* Age: Возраст человека (в годах).
* Gender: Пол человека (Male - мужчина, Female - женщина).
* Height: Рост человека (в метрах).
* Weight: Вес человека (в килограммах).
* CALC: Частота употребления алкоголя (no - не употребляет, Sometimes - иногда, Frequently - часто).
* FAVC: Частота употребления высококалорийной пищи (yes - да, no - нет).
* FCVC: Частота употребления овощей (шкала от 1 до 3, где 1 - редко, 3 - часто).
* NCP: Количество основных приемов пищи в день (от 1 до 4).
* SCC: Мониторинг потребления калорий (yes - да, no - нет).
* SMOKE: Курение (yes - да, no - нет).
* CH2O: Количество потребляемой воды в день (шкала от 1 до 3, где 1 - мало, 3 - много).
* family\_history\_with\_overweight: Наличие семейной истории избыточного веса (yes - да, no - нет).
* FAF: Физическая активность (шкала от 0 до 3, где 0 - отсутствие активности, 3 - высокая активность).
* TUE: Время, проведенное за использованием электронных устройств (шкала от 0 до 2, где 0 - мало, 2 - много).
* CAEC: Частота употребления пищи между основными приемами (no - нет, Sometimes - иногда, Frequently - часто, Always - всегда).
* MTRANS: Основной способ передвижения (Automobile - автомобиль, Bike - велосипед, Motorbike - мотоцикл, Public\_Transportation - общественный транспорт, Walking - пешком).

Классификация степеней ожирения находится в столбце NObeyesdad:

* Insufficient\_Weight - недостаточный вес
* Normal\_Weight - нормальный вес
* Overweight\_Level\_I - избыточный вес уровень I
* Overweight\_Level\_II - избыточный вес уровень II
* Obesity\_Type\_I - ожирение тип I
* Obesity\_Type\_II - ожирение тип II
* Obesity\_Type\_III – ожирение тип III

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

**РЕАЛИЗАЦИЯ ПРОЕКТА**

**Этап 1. Подготовка данных к анализу**

Подключаем необходимые библиотеки:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import warnings

Загрузим данные в объект DataFrame:

df = pd.read\_csv('ObesityDataSet\_raw\_and\_data\_sinthetic.csv')

df.head()

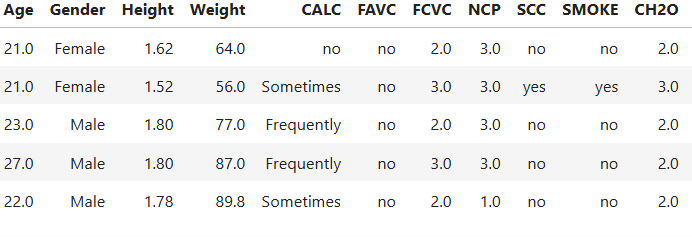


Рисунок 1. Исходный датафрейм

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

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

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

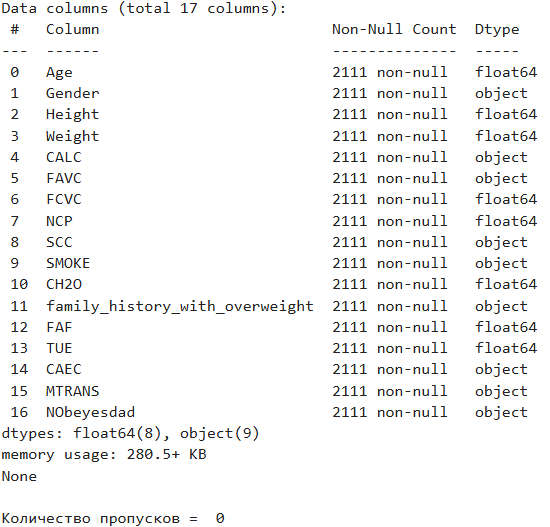


Рисунок 2. Типы данных колонок, количество пропусков

Заменяем категориальные данные числовыми:

df['FAVC'] = df['FAVC'].map({'no': 0, 'yes': 1})

df['SCC'] = df['SCC'].map({'no': 0, 'yes': 1})

df['SMOKE'] = df['SMOKE'].map({'no': 0, 'yes': 1})

df['family\_history\_with\_overweight'] = df['family\_history\_with\_overweight'].map({'no': 0, 'yes': 1})

df['Gender'] = df['Gender'].map({'Female': 0, 'Male': 1})

df['CALC'] = df['CALC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2})

df['CAEC'] = df['CAEC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})

df['MTRANS'] = df['MTRANS'].map({'Automobile': 0, 'Bike': 1, 'Motorbike': 2, 'Public\_Transportation': 3, 'Walking': 4})

df['NObeyesdad'] = df['NObeyesdad'].map({

'Insufficient\_Weight': 0,

'Normal\_Weight': 1,

'Overweight\_Level\_I': 2,

'Overweight\_Level\_II': 3,

'Obesity\_Type\_I': 4,

'Obesity\_Type\_II': 5

})

Результат:

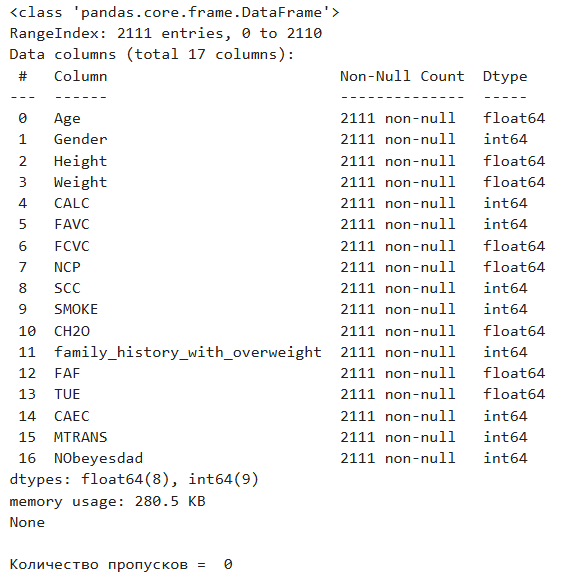


Рисунок 3. Типы данных колонок, количество пропусков

Выведем основные статистические показатели

df.describe()

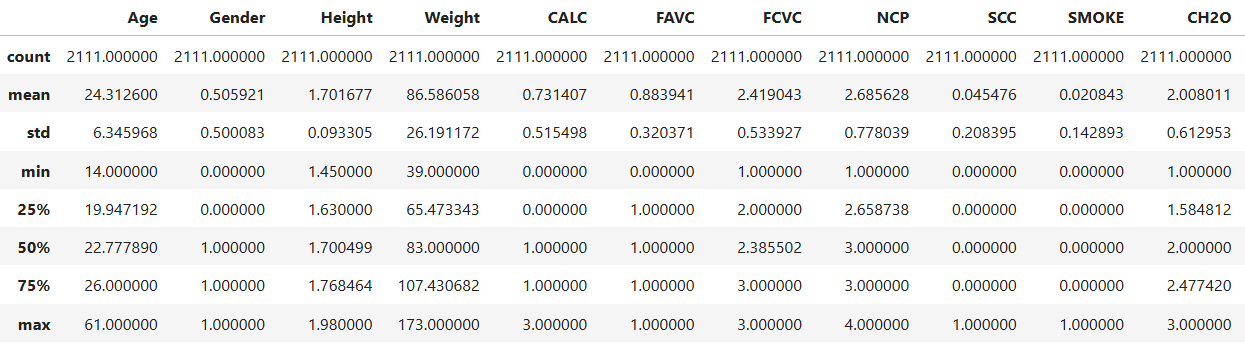


Рисунок. 4. Основные статистические показатели

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 = ['Age', 'Gender', 'Height', 'Weight', 'CALC', 'FAVC', 'FCVC', 'NCP',

'SCC', 'SMOKE', 'CH2O', 'family\_history\_with\_overweight', 'FAF', 'TUE',

'CAEC', 'MTRANS']

# Создаем экземпляр MinMaxScaler

scaler = MinMaxScaler()

# Нормализуем только выбранные столбцы

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

# Выводим результат

df.head()

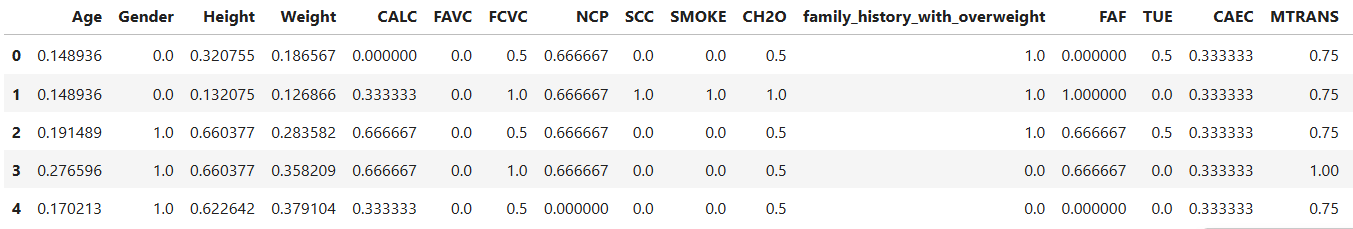


Рисунок 5. Результат нормализации

**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.

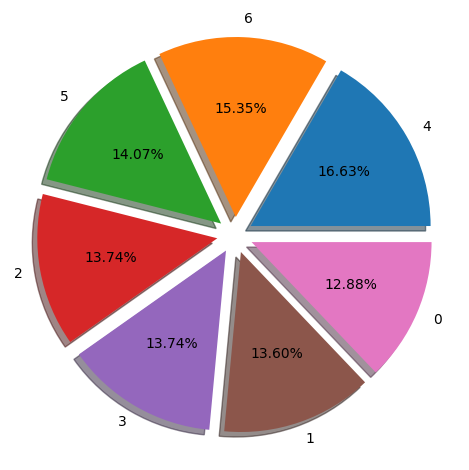


Рисунок 6. Диаграмма распределения классов

Мы видим, что классы сбалансированы.

**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

numerical\_df = df.select\_dtypes(include=['float64', 'int64'])

correlation\_matrix = numerical\_df.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

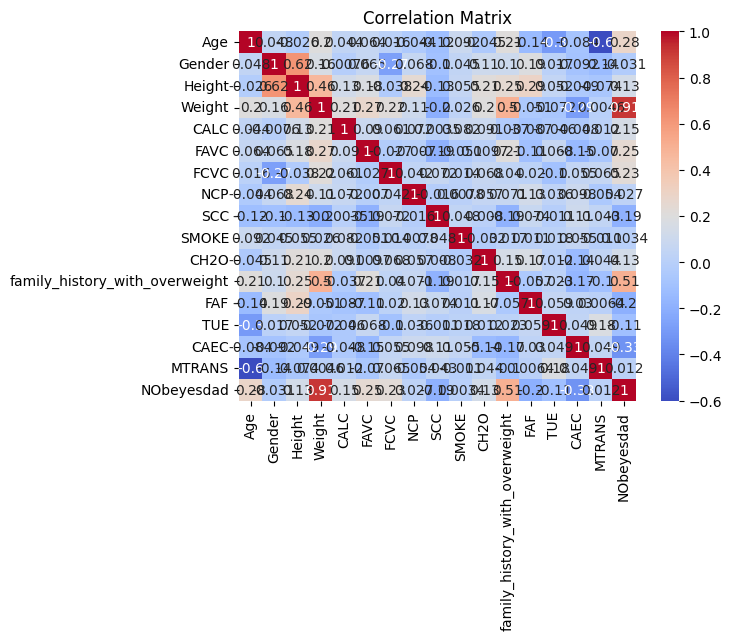


Рисунок 7. Матрица корреляции

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%

X = df.drop(columns=['NObeyesdad'])

y = df['NObeyesdad']

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 — это инструмент, предназначенный для упорядоченной обработки данных, который облегчает написание, отладку и поддержку кода, особенно в сфере машинного обучения.

import time

accuracy\_data = []

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

time1 = time.time()

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

FiPipeline = Pipeline(steps=steps)

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")

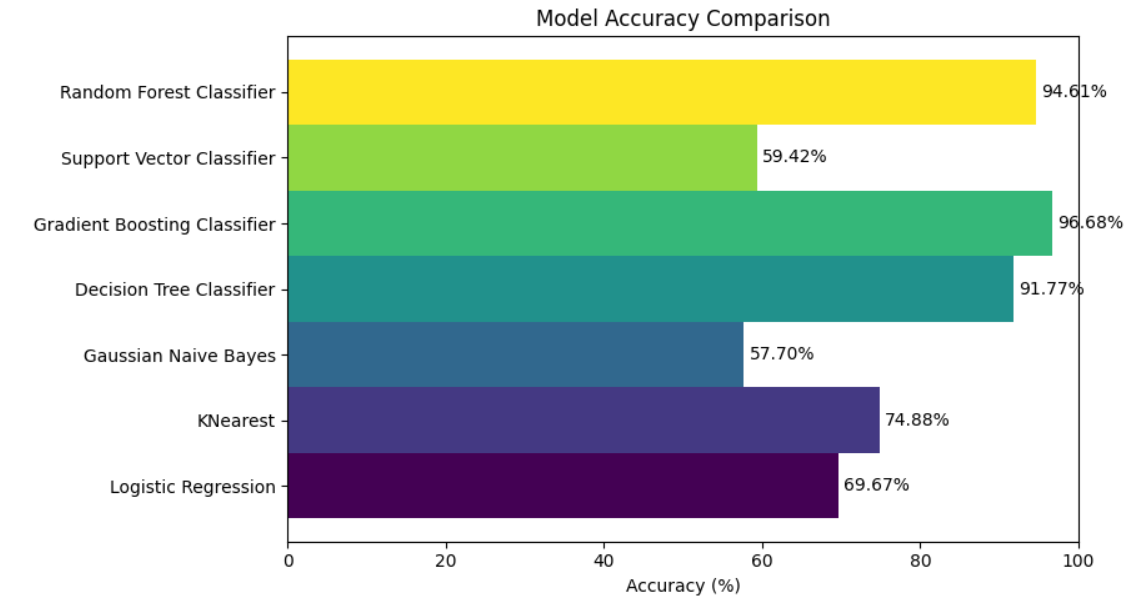


Рисунок 8. Сравнение моделей

По результатам обучения можно увидеть, что наилучшей оказалась модель Градиентного бустинга с результатом в 96%. Обучим данную модель и выведем результаты обучения.

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 = ['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)']

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

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

cm\_df = pd.DataFrame(conf\_matrix, index=['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)'], columns=['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)'])

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()

**Этап 6. Обучение модели**

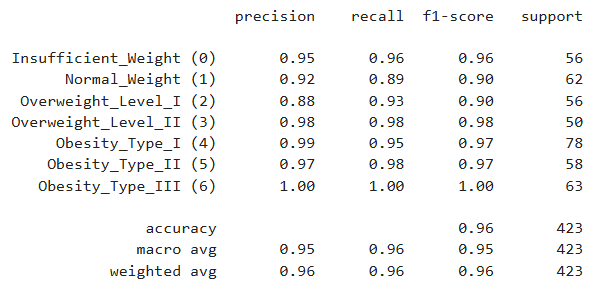


Рисунок 9. Результаты обучения модели

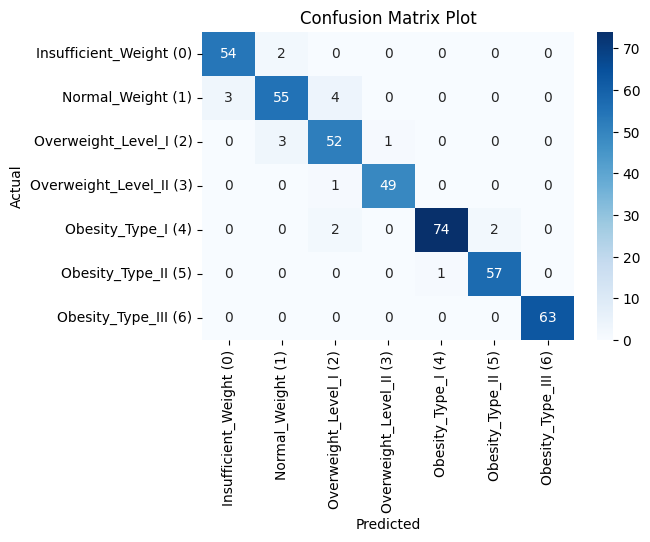
****

Рисунок 10. Результаты обучения модели

Анализируя полученные результаты, можно сделать вывод, что риск возникновения ожирения может быть предсказан с помощью модели машинного обучения. Внедрение машинного обучения в процесс анализа факторов, влияющих на развитие ожирения, значительно упрощает и ускоряет процесс. Модели, основанные на анализе данных, могут учитывать множество факторов, включая возраст, пол, индекс массы тела (ИМТ), уровень физической активности, пищевые привычки, генетическую предрасположенность и метаболические показатели, что повышает точность прогнозирования.

Большинство моделей машинного обучения продемонстрировали схожий уровень точности, однако наивысшие результаты были показаны моделью градиентного бустинга. Таким образом, поставленная цель была достигнута: на основе анализа данных разработаны модели классификации, способные предсказать риск возникновения ожирения. Однако у моделей возникают трудности в точном определении вклада каждого фактора в развитие заболевания.

Построенная модель может быть применена для анализа рисков возникновения ожирения, но с учетом имеющихся ограничений. Для повышения точности прогнозирования требуется дальнейшая работа по улучшению модели, включая сбор более детализированных данных и учет дополнительных факторов, таких как психологические аспекты и социальные условия.

**СПИСОК ЛИТЕРАТУРЫ**

1. Бабенко, Н. В. (2018). Методы машинного обучения: от теории к практике. М.: Издательство МГТУ.
2. Зайцев, А. В. (2020). Основы анализа данных и машинного обучения. М.: Наука.
3. Барабанов, А. В., & Кузнецов, И. А. (2019). Введение в машинное обучение. М.: Издательство ИТМО.
4. Гусев, А. В. (2021). Машинное обучение и его применение в биоинформатике. М.: Физматлит.
5. Капитонов, А. А. (2017). Обработка и анализ данных с использованием Python и R. М.: БХВ-Петербург.
6. Сидоренко, В. В. (2019). Классификация и регрессия в машинном обучении: теоретические аспекты и практические применения. М.: ЛКИ.
7. Ширяев, А. Н. (2016). Статистические методы и их применение в машинном обучении. М.: Издательство МГУ.

**Приложения**

Приложение 1

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

from imblearn.pipeline import Pipeline

from imblearn.under\_sampling import RandomUnderSampler

df = pd.read\_csv('ObesityDataSet\_raw\_and\_data\_sinthetic.csv')

df.head()

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

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

df['FAVC'] = df['FAVC'].map({'no': 0, 'yes': 1})

df['SCC'] = df['SCC'].map({'no': 0, 'yes': 1})

df['SMOKE'] = df['SMOKE'].map({'no': 0, 'yes': 1})

df['family\_history\_with\_overweight'] = df['family\_history\_with\_overweight'].map({'no': 0, 'yes': 1})

df['Gender'] = df['Gender'].map({'Female': 0, 'Male': 1})

df['CALC'] = df['CALC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})

df['CAEC'] = df['CAEC'].map({'no': 0, 'Sometimes': 1, 'Frequently': 2, 'Always': 3})

df['MTRANS'] = df['MTRANS'].map({'Automobile': 0, 'Bike': 1, 'Motorbike': 2, 'Public\_Transportation': 3, 'Walking': 4})

df['NObeyesdad'] = df['NObeyesdad'].map({

'Insufficient\_Weight': 0,

'Normal\_Weight': 1,

'Overweight\_Level\_I': 2,

'Overweight\_Level\_II': 3,

'Obesity\_Type\_I': 4,

'Obesity\_Type\_II': 5,

'Obesity\_Type\_III': 6

})

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

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

df.describe()

df.columns

from sklearn.preprocessing import MinMaxScaler

columns\_to\_normalize = ['Age', 'Gender', 'Height', 'Weight', 'CALC', 'FAVC', 'FCVC', 'NCP',

'SCC', 'SMOKE', 'CH2O', 'family\_history\_with\_overweight', 'FAF', 'TUE',

'CAEC', 'MTRANS']

# Создаем экземпляр MinMaxScaler

scaler = MinMaxScaler()

# Нормализуем только выбранные столбцы

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

# Выводим результат

df.head()

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

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

plt.tight\_layout()

plt.axis('off');

import seaborn as sns

numerical\_df = df.select\_dtypes(include=['float64', 'int64'])

correlation\_matrix = numerical\_df.corr()

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm')

plt.title('Correlation Matrix')

plt.show()

from sklearn.calibration import LabelEncoder

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

X = df.drop(columns=['NObeyesdad'])

y = df['NObeyesdad']

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()

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),

}

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")

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 = ['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)']

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

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

cm\_df = pd.DataFrame(conf\_matrix, index=['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)'], columns=['Insufficient\_Weight (0)', 'Normal\_Weight (1)', 'Overweight\_Level\_I (2)', 'Overweight\_Level\_II (3)', 'Obesity\_Type\_I (4)', 'Obesity\_Type\_II (5)', 'Obesity\_Type\_III (6)'])

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()