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 «Прикладной анализ данных»

**Классификация и прогнозирование диапазона цен на мобильные телефоны**

Разработчики проекта:

Колядин И. С.

Жаббаров К. Р.

Леменко Д. А.

Степанов С.В.

Пермь, 2024

**Оглавление**

[ПАСПОРТ ПРОЕКТА 3](#_Toc185538135)

[СОДЕРЖАНИЕ ПРОЕКТА 5](#_Toc185538136)

[Анализ проблемы исследования 5](#_Toc185538137)

[Исходные данные 7](#_Toc185538138)

[Реализация проекта 9](#_Toc185538139)

[Этап 2. Предварительный анализ данных 11](#_Toc185538140)

[Этап 3. Корреляционный анализ данных 13](#_Toc185538141)

[Этап 4. Моделирование 14](#_Toc185538142)

[4.1 Метод опорных векторов 14](#_Toc185538143)

[4.2 Дерево решений 18](#_Toc185538144)

[4.3 Случайный лес 20](#_Toc185538145)

[Этап 5. Прогнозирование 22](#_Toc185538146)

[Заключение 24](#_Toc185538147)

[Список использованных источников и литературы 25](#_Toc185538148)

[Приложения 26](#_Toc185538149)

ПАСПОРТ ПРОЕКТА

**Название проекта:**Классификация и прогнозирование диапазона цен на мобильные телефоны.

**Сведения об авторах:**Колядин Иван Сергеевич, Жаббаров Ким Рустемович, Леменко Дмитрий Андреевич, Степанов Степан Владимирович.

**Цель:**разработка модели, позволяющей классифицировать смартфоны по ценовым сегментам и прогнозировать стоимость устройств на основе их технических характеристик.

**Задачи:**

1. Провести обзор проблемы и обосновать ее актуальность в контексте современных тенденций рынка мобильных устройств.
2. Загрузить и подготовить данные для анализа, включая устранение пропусков и аномалий.
3. Выполнить предварительный анализ, включая выявление взаимосвязей между характеристиками и ценой, а также обработку выбросов.
4. Создать и протестировать модели классификации, применяя методы машинного обучения.
5. Провести прогнозирование ценового диапазона для новых устройств и оценить точность моделей.

**Краткое описание проекта:**

Требуется проанализировать данные по мобильным телефонам и определить, возможно ли распределить телефоны, по классам в зависимости от схожести атрибутов. Дать интерпретацию полученным результатам. Сделать выводы.

**Конкретные ожидаемые результаты:**

Построенная классификационная модель и рекомендации по ее использованию.

СОДЕРЖАНИЕ ПРОЕКТА

Анализ проблемы исследования

Анализ классификации и прогнозирования диапазона цен на мобильные телефоны в зависимости от их характеристик является важным направлением исследования потребительского рынка.

Характеристики устройств, такие как объем оперативной памяти, емкость аккумулятора, размер дисплея, производительность процессора и качество камеры, оказывают значительное влияние на ценообразование. Понимание этой зависимости позволяет не только оценивать текущие рыночные тенденции, но и предсказывать ценовые категории для новых моделей. Это актуально как для производителей, стремящихся адаптироваться к запросам аудитории, так и для покупателей, желающих выбирать устройства с оптимальным соотношением цены и качества.

На современном рынке мобильных устройств диапазоны цен значительно варьируются в зависимости от указанных характеристик. Анализ ключевых характеристик и их влияния на стоимость позволяет классифицировать смартфоны на категории и предсказывать ценовые диапазоны для моделей, которые еще не поступили в продажу.

Проведение такого анализа требует применения количественных методов, включающих обработку большого массива данных. На этапе предварительного анализа необходимо устранить пропущенные значения, выявить выбросы, проверить распределение данных на нормальность и оценить степень их корреляции.

*Целью исследования* является разработка модели, позволяющей классифицировать смартфоны по ценовым сегментам и прогнозировать стоимость устройств на основе их технических характеристик.

*Задачи:*

1. Провести обзор проблемы и обосновать ее актуальность в контексте современных тенденций рынка мобильных устройств.
2. Загрузить и подготовить данные для анализа, включая устранение пропусков и аномалий.
3. Выполнить предварительный анализ, включая выявление взаимосвязей между характеристиками и ценой, а также обработку выбросов.
4. Создать и протестировать модели классификации, применяя методы машинного обучения.
5. Провести прогнозирование ценового диапазона для новых устройств и оценить точность моделей.

Исходные данные

Список колонок анализируемого набора данных:

* **id** - ID
* **battery\_power** - Общая энергия, которую может накопить аккумулятор за один раз, измеряется в мАч.
* **blue** - Наличие Bluetooth.
* **clock\_speed** - Скорость, с которой микропроцессор выполняет команды.
* **dual\_sim** - Поддержка двух sim-карт.
* **fc** – Количество мегапикселей фронтальной камеры.
* **four\_g** – Наличие 4G.
* **int\_memory** - Объем встроенной памяти в гигабайтах.
* **m\_dep** - Глубина в см.
* **mobile\_wt** – Вес.
* **n\_cores** - Количество ядер процессора.
* **pc** – Количество мегапикселей основной камеры.
* **px\_height** - Высота разрешения в пикселях.
* **px\_width** - Ширина разрешения в пикселях.
* **ram** – Количество оперативной памяти в мегабайтах.
* **sc\_h** - Высота экрана в см.
* **sc\_w** - Ширина экрана в см.
* **talk\_time** - Длительность работы на одним заряде аккумулятора.
* **three\_g** – Наличие 3G.
* **touch\_screen** – Налчиче сенсорного экрана.
* **wifi** – Налчичие wifi.
* **price\_range** – Цена телефона:
  + 0 (низкая)
  + 1 (средняя)
  + 2 (высокая)
  + 3 (очень высокая)

Необходимо проанализировать данные по мобильным телефонам и определить, возможно ли распределить телефоны, по классам в зависимости от схожести атрибутов.

Выдвинем гипотезу исследования: Диапазон цен на мобильные телефоны зависит от их характеристик и может быть смоделирован с применением моделей классификации.

Реализация проекта

**Этап 1. Подготовка данных к анализу**

Подключим необходимые библиотеки:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

import warnings

import plotly.graph\_objs as go

from plotly.subplots import make\_subplots

from matplotlib import colors

from matplotlib.colors import ListedColormap, LinearSegmentedColormap

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score, roc\_auc\_score

from sklearn.metrics import classification\_report, RocCurveDisplay, ConfusionMatrixDisplay

from sklearn.base import clone

from sklearn.impute import KNNImputer

from sklearn.pipeline import Pipeline

from plotly.offline import init\_notebook\_mode

init\_notebook\_mode(connected=True)

warnings.filterwarnings('ignore')

Загрузим датафрейм:

df = pd.read\_csv('train.csv')

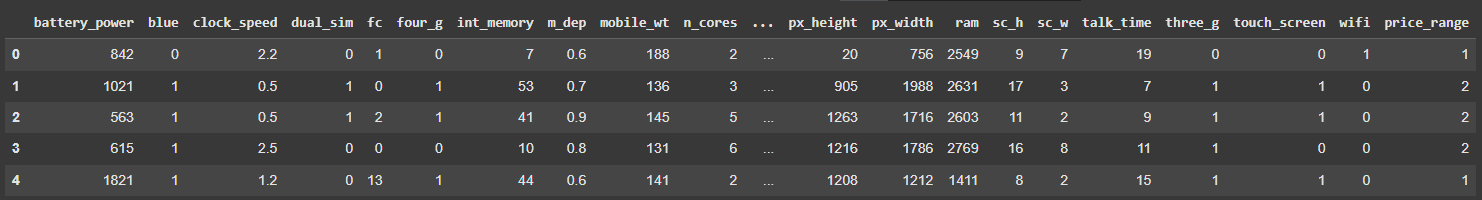
df.head()

Рисунок 1. Исходный датафрейм

Убедимся, что все столбцы имеют числовой тип.

df.info()

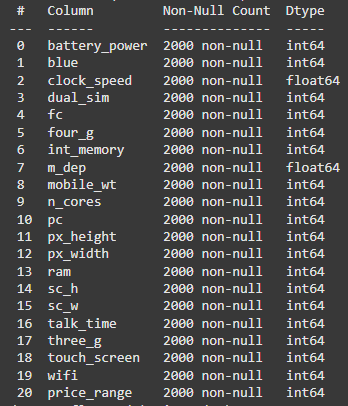


Рисунок 2. Типы данных колонок

Все колонки обладают числовым типом.

Проверим данные на пропуски и дубликаты. Если в столбцах или строках очень много пропусков, такие признаки и объекты для анализа малоинформативны, их следует удалить. Посмотрим, количество дубликатов в датафрейме и количество пропусков по колонкам:

df.duplicated().sum()

0

df.isnull().sum()

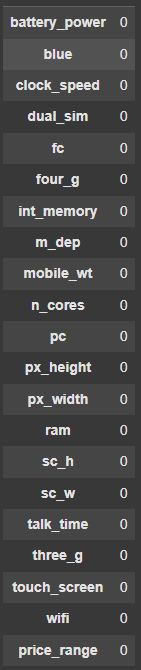


Рисунок 3. Количество пустых значений в столбцах

Итак, результат первого этапа – это готовый к анализу набор данных в виде датафрейма.

Этап 2. Предварительный анализ данных

***Аномальными наблюдениями*** (*выбросами*, англ. *Outliers, Extreme values*) называют такие значения уровня временного ряда, которые значительно отличаются от остальных. При выявлении подобных «выбросов» возникают серьезные вопросы: являются ли отклоняющиеся данные действительно ошибками (например, регистрации) или это реальные значения и как получить адекватные оценки для параметров изучаемой совокупности.

Рассмотрим на распределение столбцов px\_height и sc\_w:



Рисунок 4. Распределение столбцов px\_height и sc\_w

Видно, что данные столбцы содержат много значений, равных нулю или близких к нулю, которые выглядят как шум, но поскольку нет достаточных знаний в этой области, невозможно установить определенный нижний предел для этих колонок.

В интернете есть информация, что некоторые из самых ранних серийно выпускаемых мобильных телефонов, имели разрешение экрана всего в 5 пикселей по высоте, а ширину около 2-3 сантиметров.

Далее модель будет обучена методом оперных векторов. Одно из его преимуществ заключается в том, что он менее чувствителен к шуму и выбросам в данных по сравнению с другими алгоритмами, такими как деревья решений (который тоже будет рассмотрен).

Поэтому рассматриваем значение 5 для px\_height и значение 2 для sc\_w как минимально допустимые значения и определяем индексы выборок шума:

sc\_w\_NoiseIndex = df[df['sc\_w'] < 2].index

px\_height\_NoiseIndex = df[df['px\_height'] < 5].index

Были получены следующие результаты:

* Количество значений, которые определяем как шум для sc\_w: 390.
* Количество значений, которые определяем как шум для px\_height: 9.

На данном этапе не предпринимаем действий для их очистки. Вернемся к этим данным, когда достигнем чувствительную к шуму модели, как дерево решений.

Этап 3. Корреляционный анализ данных

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

Выполним корреляционный анализ данных с помощью матрицы корреляции. Матрицу корреляции отобразим с помощью диаграммы «тепловая карта» (heatmap).

target = 'price\_range'

df\_ordered = pd.concat([df.drop(target,axis=1), df[target]],axis=1)

corr = df\_ordered.corr()

plt.figure(figsize=(18,12))

sns.heatmap(corr, annot=True, fmt='.2f', linewidths=0.2)

plt.show()

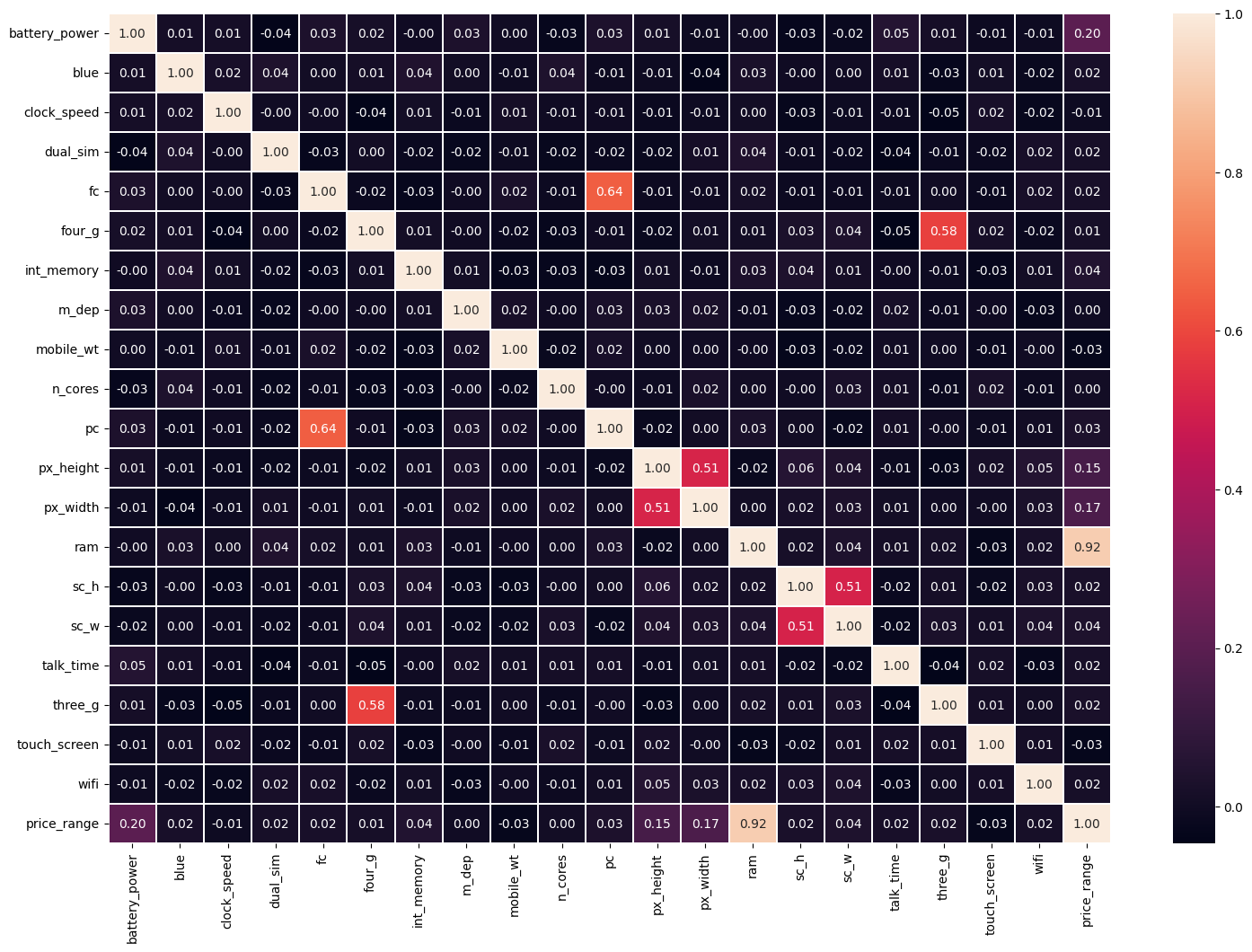


Рисунок 4. Тепловая карта матрицы корреляции

По результатам корреляционного анализа, видно, что наибольшее влияние на целевой показатель оказывает столбец ram. Остальные колонки имеют низкое значение корреляции с целевой переменой. Поскольку корреляция Пирсона выражает только линейную связь между двумя переменными, можно предположить, что отношения между переменными нелинейны.

Таким образом, на третьем этапе была выдвинута гипотеза о том, что диапазон цен на мобильные телефоны может быть смоделирован с применением моделей классификации.

Этап 4. Моделирование

4.1 Метод опорных векторов

Разобьём данные на обучающую и тестовую выборки:

X = df.drop('price\_range', axis=1)

y = df['price\_range']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0, stratify=y)

Масштабируем данные:

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

X\_train = pd.DataFrame(X\_train\_scaled, columns=X\_train.columns)

X\_test = pd.DataFrame(X\_test\_scaled, columns=X\_test.columns)

Определим функцию, которая найдет оптимальную комбинацию гиперпараметров для достижения максимальной точности классификатора. Функция использует решетчатый подход (GridSearchCV) и перекрестную проверку (StratifiedKFold) для оценки различных комбинаций гиперпараметров и выбирает комбинацию с наивысшей точностью. Функция возвращает лучший классификатор с оптимальными гиперпараметрами:

    cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=0)

    clf\_grid = GridSearchCV(clf, param\_grid, cv=cv, scoring='accuracy', n\_jobs=-1)

    clf\_grid.fit(X\_train, y\_train)

    print("Best hyperparameters:\n", clf\_grid.best\_params\_)

    return clf\_grid.best\_estimator\_

Настройка гиперпараметров для опорных векторов (SVM) может занять много времени, поскольку она включает в себя многократное обучение модели с разными значениями гиперпараметров и оценку ее производительности для поиска оптимального набора. После однократной настройки гиперпараметров SVM выявлено, что линейное ядро ​​является оптимальным для этого набора данных. Поэтому ограничиваем param\_grid линейным ядром, чтобы сократить время выполнения программы:

param\_grid = {'kernel': ['linear'],

               'C': [ 0.1, 1, 10, 50, 100]

             }

Найдем лучшие гиперпараметры:

svm = SVC(probability=True, random\_state=0)

svm\_optimized = tune\_hyperparameters(svm, param\_grid, X\_train, y\_train)

Вывод:

Best hyperparameters:

{'C': 50, 'kernel': 'linear'}

Важным шагом при создании хорошей модели является правильный выбор метрики для оценки её качества, поскольку неправильный выбор может привести к неверным выводам. Наиболее популярные метрики для задач классификации:

**Accuracy**

Самым простым способом оценить качество модели является Accuracy или точность, которая отражает долю правильно спрогнозированных классов среди всех образцов.

**Precision**

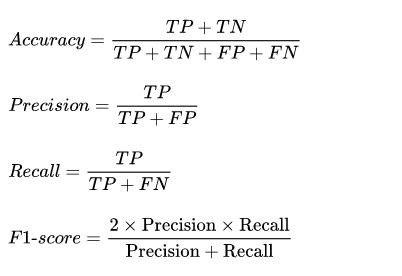
Характеризует долю правильно предсказанных положительных классов среди всех образцов, которые модель спрогнозировала как положительный класс:

**Recall**

Ещё известное как True Positive Rate, отражает долю правильно предсказанных положительных классов среди всех реальных положительных образцов.

**F1-score**

Представляет собой гармоническое среднее между Precision и Recall, обеспечивая между ними баланс, что особенно полезно при неравномерном распределении классов.



TP – положительные классы

TN отрицательные классы

FP ошибки первого рода

FN ошибки второго рода

Определим функцию для вывода оценок качества модели:

y\_pred = clf.predict(X\_test)

result = pd.DataFrame(data=[accuracy\_score(y\_test, y\_pred),

precision\_score(y\_test, y\_pred, average='macro'), recall\_score(y\_test, y\_pred, average='macro'),

f1\_score(y\_test, y\_pred, average='macro')],

index=['Accuracy','Macro Precision','Macro Recall','Macro F1-score'],columns = [model\_name])

result = (result \* 100).round(2).astype(str) + '%'

return result

Результаты оформим в виде таблицы и матрицы ошибок:



Рисунок 5. Оценка модели опорных векторов

Классификатора показал замечательную точность 98,25%, хотя набор данных был зашумлен. Если рассмотреть матрицу ошибок модели, то из 400 мобильных телефонов модель допустила ошибки только в 7 случаях, чтобы предсказать их ценовой диапазон.

4.2 Дерево решений

Дерево решений — это алгоритм, чувствительный к шуму и склонный к переобучению, особенно при обучении на зашумленных данных. Поэтому необходимо очистить набор данных. Поскольку количество шума, в наборе данных велико, нельзя удалить соответствующие строки из набора данных. Поэтому используем KNNImputer для расчета этих значений шума.

KNN Imputer — это метод, используемый для заполнения пропущенных значений в наборе данных с использованием алкоритма k-ближайших соседей. Этот метод включает в себя поиск k-ближайших соседей для точки данных с пропущенным значением и вставку пропущенного значения с использованием среднего или медианного значения соседних точек.

df\_denoised = df.copy()

# Замена шума значением nan

df\_denoised.loc[sc\_w\_NoiseIndex,'sc\_w'] = np.nan

df\_denoised.loc[px\_height\_NoiseIndex,'px\_height'] = np.nan

imputer = KNNImputer(n\_neighbors=2)

imputed\_data = imputer.fit\_transform(df\_denoised)

df\_denoised = pd.DataFrame(imputed\_data, columns=df\_denoised.columns)

X\_denoised = df\_denoised.drop('price\_range', axis=1)

df\_denoised['price\_range'] = df\_denoised['price\_range'].astype(int)

y = df\_denoised['price\_range']

Настройка гиперпараметров имеет решающее значение для деревьев решений, поскольку они склонны к переобучению. Переобучение происходит, когда дерево слишком сложное и слишком близко соответствует обучающим данным, улавливая даже шум в данных. Настройка гиперпараметров может помочь избежать переобучения и повысить производительность модели на новых данных.

**Наиболее распространенные гиперпараметры:**

* Criterion
* Maximum depth
* Minimum samples per split
* Minimum samples per leaf
* Maximum features

Существует ряд колонок, которые не оказывают большого влияния на целевую оценку. Удалив ненужные функции из набора данных с помощью методов выбора функций, мы можем упростить наше дерево решений и уменьшить его сложность, что может привести к повышению точности модели за счет уменьшения переобучения модели.

Разобьём данные на обучающую и тестовую выборки:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_denoised, y, test\_size=0.2, random\_state=0, stratify=y)

Найдем лучшие гиперпараметры:

param\_grid = {'criterion': ['gini', 'entropy', 'log\_loss'],

              'max\_depth': np.arange(1, 15),

              'min\_samples\_split': np.arange(2, 10),

              'min\_samples\_leaf': np.arange(1, 10),

              'max\_features': [None, 'sqrt', 'log2']}

dt = DecisionTreeClassifier(random\_state=0)

dt\_optimized = tune\_hyperparameters(dt, param\_grid, X\_train, y\_train)

Оценка модели:

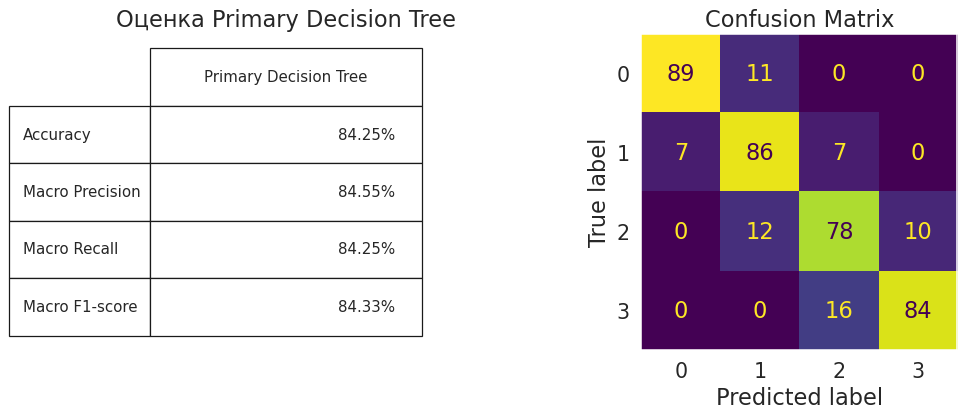


Рисунок 6. Оценка модели дерева решений

Данная модель показала себя хуже метода опорных векторов.

4.3 Случайный лес

Оптимизация гиперпараметров случайного леса может повысить точность модели и предотвратить переобучение или недостаточный подбор данных. Модель случайного леса состоит из нескольких деревьев решений, которые могут быть как поверхностными, так и глубокими. Мелкие деревья имеют ограниченное количество ветвей или уровней. в то время как глубокие деревья имеют много ветвей или уровней и не полностью выросли. Глубокие деревья, как правило, имеют низкую предвзятость, но высокую дисперсию, что делает их подходящими для методов объединения, направленных на уменьшение дисперсии. При выборе гиперпараметров для случайного леса важно учитывать наличие глубоких деревьев при выборе диапазона значений для гиперпараметров случайного леса.

**Наиболее распространенные гиперпараметры**

* n\_estimators
* criterion
* max\_depth
* min\_samples\_split
* min\_samples\_leaf
* bootstrap
* max\_features

Разобьём данные на обучающую и тестовую выборки:

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_denoised, y, test\_size=0.2, random\_state=0, stratify=y)

Найдем лучшие гиперпараметры:

param\_grid = {

    'n\_estimators': [70, 100, 120, 150],

    'max\_depth': np.arange(10, 18),

    'min\_samples\_split': [2, 3, 4],

    'min\_samples\_leaf': [1, 2, 3]

}

rf = RandomForestClassifier(criterion='gini', max\_features=None, bootstrap=True, random\_state=0)

rf\_optimised = tune\_hyperparameters(rf, param\_grid, X\_train, y\_train)

Оценка модели:

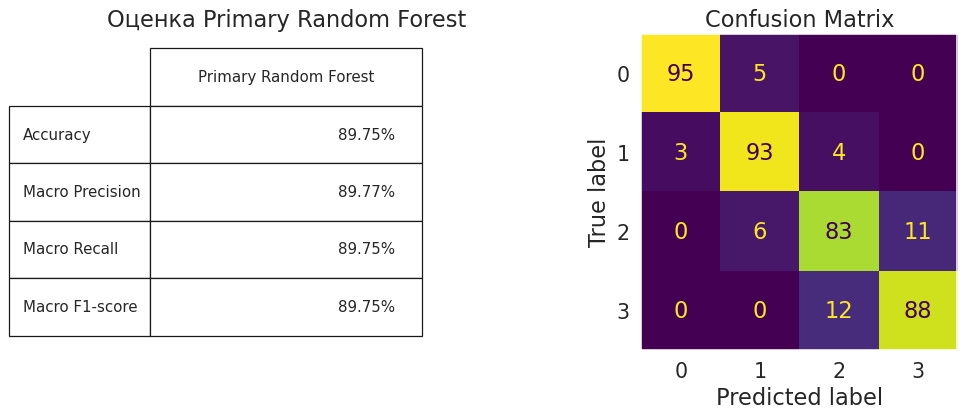


Рисунок 7. Оценка модели случайного леса

Как видно по результатам оценок случайный лес более устойчивый к шуму, чем дерево решений. Однако лучше всего себя показала модель опорных векторов.

Этап 5. Прогнозирование

Выполним прогнозирование целевой переменной на тестовой выборке:

X = df.drop('price\_range', axis=1)

y = df['price\_range']

svm\_pipeline = Pipeline([

                         ('scaler', StandardScaler()),

                         ('svm', SVC(probability=True, C=50, kernel='linear', random\_state=0))

                        ])

svm\_pipeline.fit(X, y)

df\_samples = pd.read\_csv('test.csv')

df\_samples.drop('id', axis=1, inplace=True)

df\_samples['price\_range\_pred'] = svm\_pipeline.predict(df\_samples)

df\_samples['price\_range\_pred']

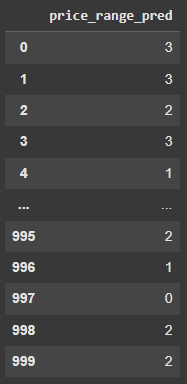


Рисунок 8. Результат прогнозирования

Визуализируем полученные данные:

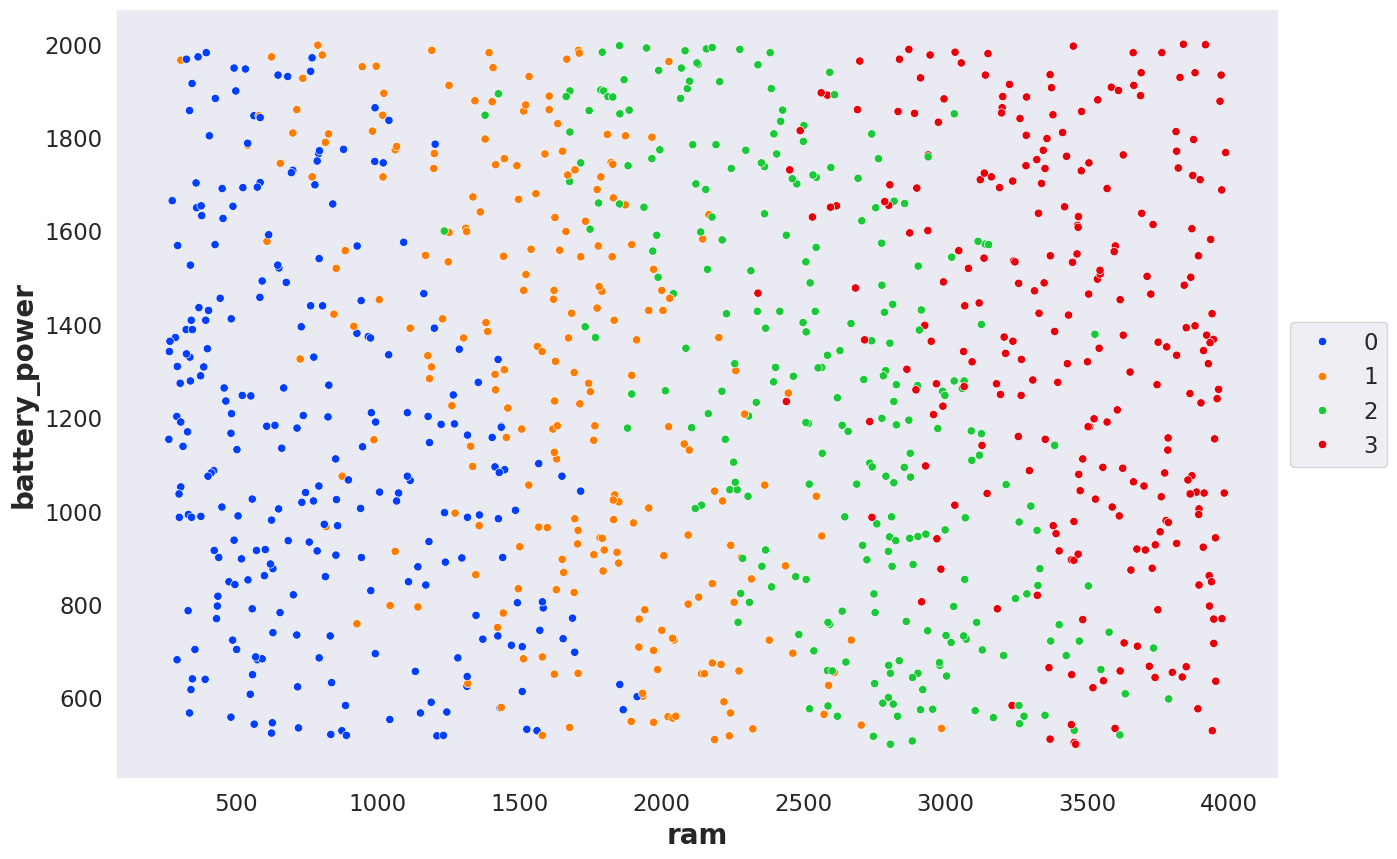


Рисунок 9 . Визуализация классификации диапазона цен

Заключение

На основе проведенного анализа можно сделать вывод, что выдвинутая гипотеза подтвердилась: диапазон цен на мобильные телефоны зависит от их характеристик и может быть смоделирован с применением моделей классификации.

Как и ожидалось, метод опорных векторов показал лучшую производительность, чем алгоритмы дерева решений и основанные на нем алгоритмы, такие как случайный лес, из-за его меньшей чувствительности к зашумленным данным, и, наконец, мы получили отличную точность для тестовых данных.

Таким образом, поставленная цель была достигнута. Был выполнен анализ данных о мобильных телефонах и построены три модели классификации, позволяющие делать прогнозы с высокой точностью.

Для достижения цели были решены следующие задачи:

* Выполнен анализ проблемы, обоснована ее актуальность.
* Осуществлена загрузка данных и подготовка их к анализу количественными методами, включая устранение пропущенных значений.
* Выполнен предварительный анализ данных, в том числе выявление и обработка выбросов, корреляционный анализ.
* Осуществлено моделирование зависимости целевого признака от факторных методом различными методами, в том числе подобрана наилучшая модель, оценено ее качество и выполнено прогнозирование.

Построенная классификационная модель может применяться для прогнозирования цен на мобильные телефоны, что может быть полезно на современном рынке мобильных устройств.

Список использованных источников и литературы

1. Андрей Бурков Машинное обучение без лишних слов. — СПб.: Питер, 2020 — 192 с.: ил. — (Серия «Библиотека программиста»).
2. Петер Флах Машинное обучение. Наука и искусство построения алгоритмов, которые извлекают знания из данных /П. Флах; пер с англ. А. А. Слинкина. 2-е изд., 401с. – Москва: ДМК Пресс, 2023
3. Вьюгин В.В. “Математические основы машинного обучения и прогнозирования” М.: 2013, 2018. - 484 с.
4. Саммерфилд М., Python на практике [Электронный ресурс] / Марк Саммерфилд - М. : ДМК Пресс, 2014. - 338 с. - ISBN 978-5-97060-095-5 - Режим доступа: http://www.studentlibrary.ru/book/ISBN9785970600955.html

Приложения

Приложение 1

Программный код

\*\*Столбцы набора данных:\*\*

- \*\*id\*\* - Идентификатор

- \*\*battery\_power\*\* - Общая энергия, которую батарея может накопить за один раз, измеряется в мАч

- \*\*blue\*\* - Есть Bluetooth или нет

- \*\*тактовая скорость\*\* - Скорость, с которой микропроцессор выполняет команды

- \*\*dual\_sim\*\* - Поддерживает две sim-карты или нет

- \*\*fc\*\* - Мегапиксельная фронтальная камера

- \*\*four\_g\*\* - Поддерживает 4G или нет

- \*\*int\_memory\*\* - Внутренняя память в гигабайтах

- \*\*m\_dep\*\* - Глубина перемещения в сантиметрах

- \*\*mobile\_wt\*\* - Вес мобильного телефона

- \*\*n\_cores\*\* - Количество ядер процессора

- \*\*pc\*\* - Мегапиксели основной камеры

- \*\*px\_height\*\* - Высота разрешения в пикселях

- \*\*px\_width\*\* - Ширина разрешения в пикселях

- \*\*ram\*\* - Оперативная память в мегабайтах

- \*\*sc\_h\*\* - Высота экрана мобильного устройства в см

- \*\*sc\_w\*\* - Ширина экрана мобильного устройства в см

- \*\*talk\_time\*\* - Максимальное время, на которое хватит одного заряда аккумулятора, когда вы находитесь в

- \*\*three\_g\*\* - Есть 3G или нет

- \*\*touch\_screen\*\* - Есть сенсорный экран или нет

- \*\*wifi\*\* - Есть Wi-Fi или нет

- \*\*price\_range\*\* - Целевая переменная со значением:

- 0 (низкая стоимость)

- 1 (средняя стоимость)

- 2 (высокая стоимость)

- 3 (очень высокая стоимость)

\*\*Модели:\*\*

- Support Vector Machine (SVM)

- Decision Tree

- Random Forest

# Импортирование библиотек

"""

# Commented out IPython magic to ensure Python compatibility.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import missingno as msno

import warnings

import plotly.graph\_objs as go

from plotly.subplots import make\_subplots

from matplotlib import colors

from matplotlib.colors import ListedColormap, LinearSegmentedColormap

from sklearn.model\_selection import train\_test\_split, GridSearchCV, StratifiedKFold, cross\_val\_score

from sklearn.preprocessing import StandardScaler

from sklearn.svm import SVC

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, recall\_score, precision\_score, f1\_score, roc\_auc\_score

from sklearn.metrics import classification\_report, RocCurveDisplay, ConfusionMatrixDisplay

from sklearn.base import clone

from sklearn.impute import KNNImputer

from sklearn.pipeline import Pipeline

# %matplotlib inline

from plotly.offline import init\_notebook\_mode

init\_notebook\_mode(connected=True)

warnings.filterwarnings('ignore')

"""#Чтение данных"""

df = pd.read\_csv('train.csv')

df.head()

"""#Предобработка"""

df.info()

"""## Проверка на дупликаты и пропуски"""

df.duplicated().sum()

df.isnull().sum()

"""# Обнаружение шума"""

df\_numerical = df [['battery\_power', 'clock\_speed', 'fc', 'int\_memory', 'm\_dep',

'mobile\_wt', 'pc', 'px\_height', 'px\_width', 'ram', 'talk\_time', 'sc\_h', 'sc\_w']]

for col in df\_numerical[:-1]:

plt.figure(figsize=(10,5))

plt.hist(df[col], bins=30, color="blue", linewidth=1, edgecolor="black")

plt.title(col)

plt.tight\_layout();

"""Рассмотрим распределение px\_height и sc\_w"""

fig, ax = plt.subplots(1, 2, figsize=(15,5), dpi=120)

# px\_height:

\_, bin\_edges = np.histogram(df['px\_height'], range=(np.floor(df['px\_height'].min()), np.ceil(df['px\_height'].max())))

graph1 = sns.histplot(x='px\_height', data=df, bins=bin\_edges, color='blue', ax=ax[0])

ax[0].set\_xticks(np.round(bin\_edges,1))

for cont in graph1.containers:

graph1.bar\_label(cont)

# sc\_w:

\_, bin\_edges = np.histogram(df['sc\_w'], range=(np.floor(df['sc\_w'].min()), np.ceil(df['sc\_w'].max())))

graph2 = sns.histplot(x='sc\_w', data=df, bins=bin\_edges, color='blue', ax=ax[1])

ax[1].set\_xticks(np.round(bin\_edges,1))

for cont in graph2.containers:

graph2.bar\_label(cont)

plt.tight\_layout()

plt.show()

"""Столбцы содержат много значений, равных нулю или близких к нулю, которые выглядят как шум"""

noise\_sc\_w = df[df['sc\_w'] < 2].index

noise\_px\_height = df[df['px\_height'] < 5].index

print('Количество значений, которые определяем как шум для sc\_w: ', len(noise\_sc\_w))

print('Количество значений, которые определяем как шум для px\_height: ', len(noise\_px\_height))

"""# Выбор признаков

## Матрица корреляции

"""

corr = df.corr()

plt.figure(figsize=(18,12))

sns.heatmap(corr, annot=True, fmt='.2f', linewidths=0.2)

plt.show()

"""Price\_range сильно коррелирует с ram (0,92).

Price\_range имеет низкое значение корреляции с остальными

# SVM

"""

X = df.drop('price\_range', axis=1)

y = df['price\_range']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0, stratify=y)

"""Масштабирование"""

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

X\_train = pd.DataFrame(X\_train\_scaled, columns=X\_train.columns)

X\_test = pd.DataFrame(X\_test\_scaled, columns=X\_test.columns)

"""##Настройка гиперпараметров"""

def tune\_hyperparameters(clf, param\_grid, X\_train, y\_train):

"""

Параметры:

clf : классификатор

param\_grid: Словарь значений гиперпараметров

X\_train: тестовая выборка

y\_train: целевое значение

Функция возфращает лучшие параметры для классфиктора:

"""

# перекрестная проверка

cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=0)

# GridSearchCV

clf\_grid = GridSearchCV(clf, param\_grid, cv=cv, scoring='accuracy', n\_jobs=-1)

clf\_grid.fit(X\_train, y\_train)

print("Best hyperparameters:\n", clf\_grid.best\_params\_)

return clf\_grid.best\_estimator\_

# param\_grid = [{'kernel': ['poly'],

# 'degree': [2, 3, 4, 5],

# 'gamma': [1, 0.1, 0.01, 0.001],

# 'C': [0.01, 0.1, 1, 10, 100]},

# {'kernel': ['rbf','sigmoid'],

# 'gamma': [1, 0.1, 0.01, 0.001],

# 'C': [0.01, 0.1, 1, 10, 100]},

# {'kernel': ['linear'],

# 'C': [0.01, 0.1, 1, 10, 50, 100]}

# ]

param\_grid = {'kernel': ['linear'],

'C': [ 0.1, 1, 10, 50, 100]

}

svm = SVC(probability=True, random\_state=0)

# поиск лучших гиперпараметров

svm\_optimized = tune\_hyperparameters(svm, param\_grid, X\_train, y\_train)

"""## Оценка модели"""

def metrics\_calculator(clf, X\_test, y\_test, model\_name):

y\_pred = clf.predict(X\_test)

result = pd.DataFrame(data=[accuracy\_score(y\_test, y\_pred),

precision\_score(y\_test, y\_pred, average='macro'),

recall\_score(y\_test, y\_pred, average='macro'),

f1\_score(y\_test, y\_pred, average='macro')],

index=['Accuracy','Macro Precision','Macro Recall','Macro F1-score'],

columns = [model\_name])

result = (result \* 100).round(2).astype(str) + '%'

return result

def model\_evaluation(clf, X\_train, X\_test, y\_train, y\_test, model\_name):

sns.set(font\_scale=1.5)

# Тестовый набор

y\_pred\_test = clf.predict(X\_test)

print(classification\_report(y\_test, y\_pred\_test))

fig, (ax1, ax2) = plt.subplots(1, 2, dpi=90, figsize=(12, 5))

# Таблица с оценками

result = metrics\_calculator(clf, X\_test, y\_test, model\_name)

table = ax1.table(cellText=result.values, colLabels=result.columns,

rowLabels=result.index, loc='center')

table.scale(0.6, 4.2)

table.set\_fontsize(12)

ax1.axis('tight')

ax1.axis('off')

ax1.set\_title('Оценка {}'.format(model\_name), fontsize=18)

# Матрица ошибок

ConfusionMatrixDisplay.from\_estimator(clf, X\_test, y\_test,

colorbar=False, ax=ax2)

ax2.set\_title('Confusion Matrix')

ax2.grid(False)

plt.tight\_layout()

plt.show()

model\_evaluation(svm\_optimized, X\_train, X\_test, y\_train, y\_test, 'SVM')

svm\_result = metrics\_calculator(svm\_optimized, X\_test, y\_test, 'SVM')

"""# Decision Tree

Дерево решений — это алгоритм, чувствительный к шуму и склонный к переобучению, особенно при обучении на зашумленных данных. Поэтому нам необходимо очистить набор данных. Поскольку количество значений шума, обнаруженных в наборе данных велико, мы не можем удалить соответствующие строки из набора данных. Поэтому мы используем KNNImputer для расчета этих значений шума

"""

df\_denoised = df.copy()

# Замена шума значением nan

df\_denoised.loc[noise\_sc\_w,'sc\_w'] = np.nan

df\_denoised.loc[noise\_px\_height,'px\_height'] = np.nan

imputer = KNNImputer(n\_neighbors=2)

imputed\_data = imputer.fit\_transform(df\_denoised)

df\_denoised = pd.DataFrame(imputed\_data, columns=df\_denoised.columns)

X\_denoised = df\_denoised.drop('price\_range', axis=1)

df\_denoised['price\_range'] = df\_denoised['price\_range'].astype(int)

y = df\_denoised['price\_range']

"""# Настройка гиперпараметров"""

param\_grid = {'criterion': ['gini', 'entropy', 'log\_loss'],

'max\_depth': np.arange(1, 15),

'min\_samples\_split': np.arange(2, 10),

'min\_samples\_leaf': np.arange(1, 10),

'max\_features': [None, 'sqrt', 'log2']}

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_denoised, y, test\_size=0.2, random\_state=0, stratify=y)

dt = DecisionTreeClassifier(random\_state=0)

dt\_optimized = tune\_hyperparameters(dt, param\_grid, X\_train, y\_train)

"""## Оценка модели"""

model\_evaluation(dt\_optimized, X\_train, X\_test, y\_train, y\_test, 'Primary Decision Tree')

dt\_result = metrics\_calculator(dt\_optimized, X\_test, y\_test, 'Decision Tree')

"""# Random Forest

## Настройка гиперпараметров

Оптимизация гиперпараметров случайного леса может повысить точность модели и предотвратить переобучение или недостаточное подбор данных. Модель случайного леса состоит из нескольких деревьев решений, которые могут быть как поверхностными, так и глубокими. Мелкие деревья имеют ограниченное количество ветвей или уровней.

в то время как глубокие деревья имеют много ветвей или уровней и не полностью выросли. Глубокие деревья, как правило, имеют низкую предвзятость, но высокую дисперсию, что делает их подходящими для методов объединения, направленных на уменьшение дисперсии. При выборе гиперпараметров для случайного леса

важно учитывать наличие глубоких деревьев при выборе диапазона значений для гиперпараметров случайного леса.

\*\*Наиболее распространенные гиперпараметры\*\*

\* n\_estimators

\* criterion

\* max\_depth

\* min\_samples\_split

\* min\_samples\_leaf

\* bootstrap

\* max\_features

"""

param\_grid = {

'n\_estimators': [70, 100, 120, 150],

'max\_depth': np.arange(10, 18),

'min\_samples\_split': [2, 3, 4],

'min\_samples\_leaf': [1, 2, 3]

}

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_denoised, y, test\_size=0.2, random\_state=0, stratify=y)

rf = RandomForestClassifier(criterion='gini', max\_features=None, bootstrap=True, random\_state=0)

rf\_optimised = tune\_hyperparameters(rf, param\_grid, X\_train, y\_train)

"""## Оценка"""

model\_evaluation(rf\_optimised, X\_train, X\_test, y\_train, y\_test, 'Primary Random Forest')

# Save the final performance of Random Forest classifier

rf\_result = metrics\_calculator(rf\_optimised, X\_test, y\_test, 'Random Forest')

"""#Выводы

В этом проекте мы столкнулись с проблемой многоклассовой классификации, основной проблемой которой был зашумленный набор данных и отсутствие достаточных знаний предметной области для очистки набора данных. Как и ожидалось, алгоритм SVM имел лучшую производительность, чем алгоритмы дерева решений и случайный лес, из-за его меньшей чувствительности к зашумленным данным.

"""

# Concatenate previous classifiers perfermance results into a single dataframe

results = pd.concat([svm\_result, dt\_result, rf\_result], axis=1).T

# Sort the dataframe in descending order based on accuracy

results.sort\_values(by='Accuracy', ascending=False, inplace=True)

# Color the accuracy column

results.style.applymap(lambda x: 'background-color: red', subset='Accuracy')

"""Прогнозируем класс, используя модель SVM:"""

X = df.drop('price\_range', axis=1)

y = df['price\_range']

svm\_pipeline = Pipeline([

('scaler', StandardScaler()),

('svm', SVC(probability=True, C=50, kernel='linear', random\_state=0))

])

svm\_pipeline.fit(X, y)

df\_samples = pd.read\_csv('test.csv')

df\_samples.drop('id', axis=1, inplace=True)

df\_samples['price\_range\_pred'] = svm\_pipeline.predict(df\_samples)

df\_samples['price\_range\_pred']

plt.figure(figsize=(15,10))

sns.scatterplot(data=df\_samples, x='ram', y='battery\_power', hue='price\_range\_pred')

plt.legend(loc='upper right')

plt.show()

plt.figure(figsize=(15,10))

sns.set\_palette('bright')

sns.scatterplot(data=df\_samples,x='ram',y='battery\_power',hue='price\_range\_pred',palette='bright')

plt.legend(loc='center left',bbox\_to\_anchor=(1,0.5))

plt.xlabel("ram",fontsize=20,fontweight='bold')

plt.ylabel('battery\_power',fontsize=20,fontweight='bold')

plt.grid()

plt.show()