

**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ  
ФЕДЕРАЦИИ**

Федеральное государственное автономное образовательное учреждение высшего  
профессионального образования

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТОМСКИЙ  
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**

Инженерная школа информационных технологий и робототехники

Направление 15.04.06 «Мехатроника и робототехника»

Отделение автоматизации и робототехники

«Методы искусственного интеллекта. EDA. Линейная регрессия. Дерево  
решений. CatBoost. XGBoost. Нейронные сети (MLP)»

Отчет по лабораторной работе № 1-2

по дисциплине «Методы искусственного интеллекта в мехатронике и  
робототехнике»

Наименование учебной дисциплины

Выполнил студент гр.8ЕМ42

\_\_\_\_\_

Подпись

04.03.25

Дата

Манзаров В.С.

И.О. Фамилия

Проверил ассистент ОАР

Должность

\_\_\_\_\_

Подпись

\_\_\_\_\_

Дата

Куренко В.А.

И.О. Фамилия

Томск – 2025 г.

**Цель** данной работы заключается в получении навыков анализа первичных данных и определение признаков взаимосвязи (EDA), понимания моделей: линейная регрессия, дерево решений, CatBoost, XGBoost, нейронные сети (MLP) и умения разрабатывать программу на языке Python для реализации представленных моделей.

Задачи:

- 1) Описать датасета и определить влияние признаков и выбрать признаки, которые наиболее подходят для поставленной задачи предсказания (EDA).
- 2) Построить пайплайн (DVC) исходя из результатов EDA;
- 3) Реализовать линейную регрессию, определить веса, метрики и ошибки;
- 4) Реализовать дерево решений, определить метрики и ошибки. Привести рисунок первых узлов дерева решений;
- 5) Реализовать CatBoost, определить метрики и ошибки. Выгрузить Feature Importance;
- 6) Реализовать XGBoost, определить метрики и ошибки. Выгрузить Feature Importance;
- 7) Реализовать нейронную сети, определить метрики, ошибки, кривые обучения, гистограммы весов с интерпретацией и график из Tensorboard;
- 8) Выгрузить конечный вычислительный граф DVC;
- 9) Построить сводную таблицу с метриками и сделать вывод какая модель отработала лучше и почему;
- 10) Сделать вывод по работе.

## Ход работы

Согласно варианту 12, исходный датасет состоит двух .csv-файлов, которые содержат информацию о рейтинге пользователей, идентификационному номеру пользователей, аниме, названия, жанра, количества эпизодов, типа и общего рейтинга кинокартины.

	anime_id	name				
0	32281	Kimi no Na wa.				
1	5114	Fullmetal Alchemist: Brotherhood				
2	28977	Gintama°				
3	9253	Steins;Gate				
4	9969	Gintama&#039;				

		genre	type	episodes	rating
0		Drama, Romance, School, Supernatural	Movie	1	9.37
1	Action, Adventure, Drama, Fantasy, Magic, Mili...		TV	64	9.26
2	Action, Comedy, Historical, Parody, Samurai, S...		TV	51	9.25
3		Sci-Fi, Thriller	TV	24	9.17
4	Action, Comedy, Historical, Parody, Samurai, S...		TV	51	9.16

	members
0	200630
1	793665
2	114262
3	673572
4	151266

Рисунок 1 – Исходный датасет «anime.csv»

	user_id	anime_id	rating
0	1	20	-1
1	1	24	-1
2	1	79	-1
3	1	226	-1
4	1	355	-1

Рисунок 2 – Исходный датасет «rating.csv»

Задание в целом состоит из синтеза разных интеллектуальных моделей (линейная регрессия, древо решений, CatBoost, XGBoost и обычная нейросеть) и построения метрик для оценки точности предсказания рейтинга аниме на основе входных данных. В итоге предстоит сравнить, какая модель показывает лучшие результаты.

*Пункты 1-2: подготовка датасета*

Задание выполняется в среде разработки «PyCharm», Python версии 3.11, установленные библиотеки: «DVC», «Jupyter Notebook», «Seaborn», «Pandas», «Numpy» и «Matplotlib». Для начала исходные файлы были объединены в один общий датасет, перед этим из таблиц убраны нулевые значения (Рисунок 3).

```
[4]: print(ratings.isnull().sum())
      print(anime.isnull().sum())
```

```
user_id      0
anime_id     0
rating        0
dtype: int64
anime_id     0
name         0
genre        58
type         22
episodes      0
rating       210
members       0
dtype: int64
```

```
[5]: print(ratings["rating"].unique())
```

```
[-1 10  8  9  6  7  3  5  4  1  2]
```

```
[6]: df = ratings.merge(anime, on="anime_id", how="left")
      print(df.head())
```

	user_id	anime_id	rating_x	name \
0	1	20	-1	Naruto
1	1	24	-1	School Rumble
2	1	79	-1	Shuffle!
3	1	226	-1	Elfen Lied
4	1	355	-1	Shakugan no Shana

	genre	type	episodes	rating_y \
0	Action, Comedy, Martial Arts, Shounen, Super P...	TV	220	7.81
1	Comedy, Romance, School, Shounen	TV	26	8.06
2	Comedy, Drama, Ecchi, Fantasy, Harem, Magic, R...	TV	24	7.31
3	Action, Drama, Horror, Psychological, Romance,...	TV	13	7.85
4	Action, Drama, Fantasy, Romance, School, Super...	TV	24	7.74

	members
0	683297.0
1	178553.0
2	158772.0
3	623511.0
4	297058.0

Рисунок 3 – Объединённый датасет

Далее следует рассмотреть, сколько пользователей так и не оценили аниме, и убрать их для более «полезного» анализа (Рисунок 4).

```
sns.histplot(df["rating_x"], bins=10, kde=True)
plt.show()
```

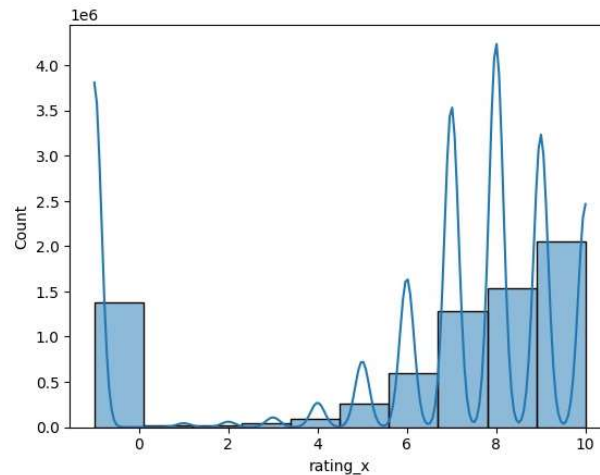


Рисунок 4 – Гистограмма распределения оценок пользователей

Как видно, виден явный перекос в сторону числа «-1», что обозначает отсутствие какой-либо оценки со стороны пользователя. Такие показатели с большей долей вероятности не дадут будущей модели ничего, только если аниме не находится в жанре картин для взрослых.

Далее оценим каждый признак отдельно. Сначала рассмотрим, как влияет на общий рейтинг тип аниме (ТВ, Сериал, Фильм и т.д.).

```
plt.figure(figsize=(12, 5))
sns.boxplot(x=df["type"], y=df["rating_x"])
plt.xticks(rotation=45)
plt.show()
```

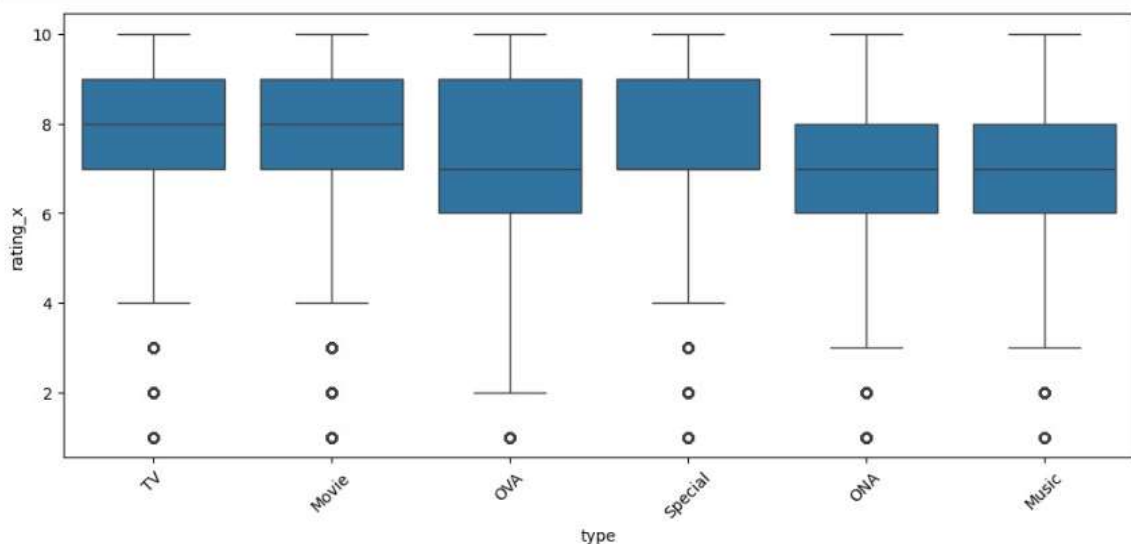


Рисунок 5 – «Ящик с усами» (boxplot) для признака – тип аниме

Следующие признаки (Рисунки 6 и 7) – число эпизодов и число фанатов (участников).

```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df["episodes"], y=df["rating_x"], alpha=0.5)
plt.xscale("log")
plt.show()
```

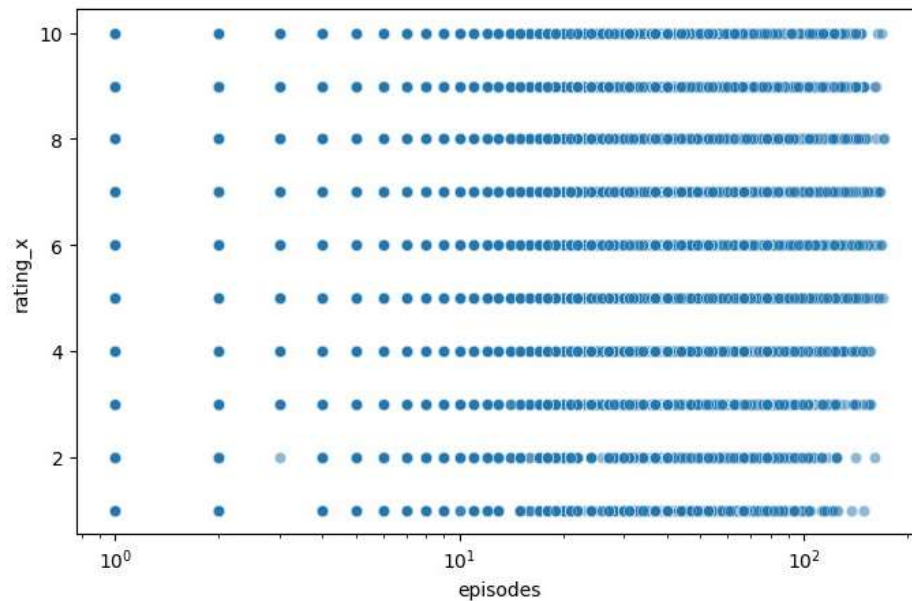


Рисунок 6 – Диаграмма рассеяния (scatterplot) для признака – эпизоды

```
plt.figure(figsize=(8, 5))
sns.scatterplot(x=df["members"], y=df["rating_x"], alpha=0.5)
plt.xscale("log")
plt.xlabel("Количество участников")
plt.ylabel("Оценка пользователя")
plt.show()
```

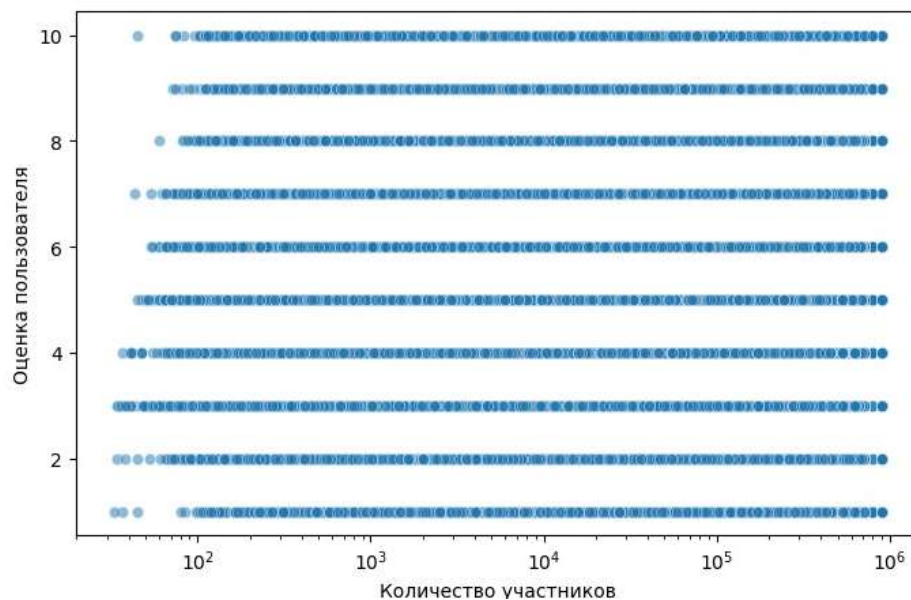


Рисунок 7 – Диаграмма рассеяния (scatterplot) для признака – фанаты

Данные диаграммы не дают полноценной картины для понимания, насколько сильно значения признаков влияют на общий рейтинг, поскольку распределение для каждой оценки практически совпадает.

Далее идет наиболее важный по моему мнению признак – жанр аниме. Для оценки влияния данного признака изначально каждый экземпляр разделяется на отдельный жанр, так как одна кинокартина может совмещать несколько жанров (Рисунок 8).

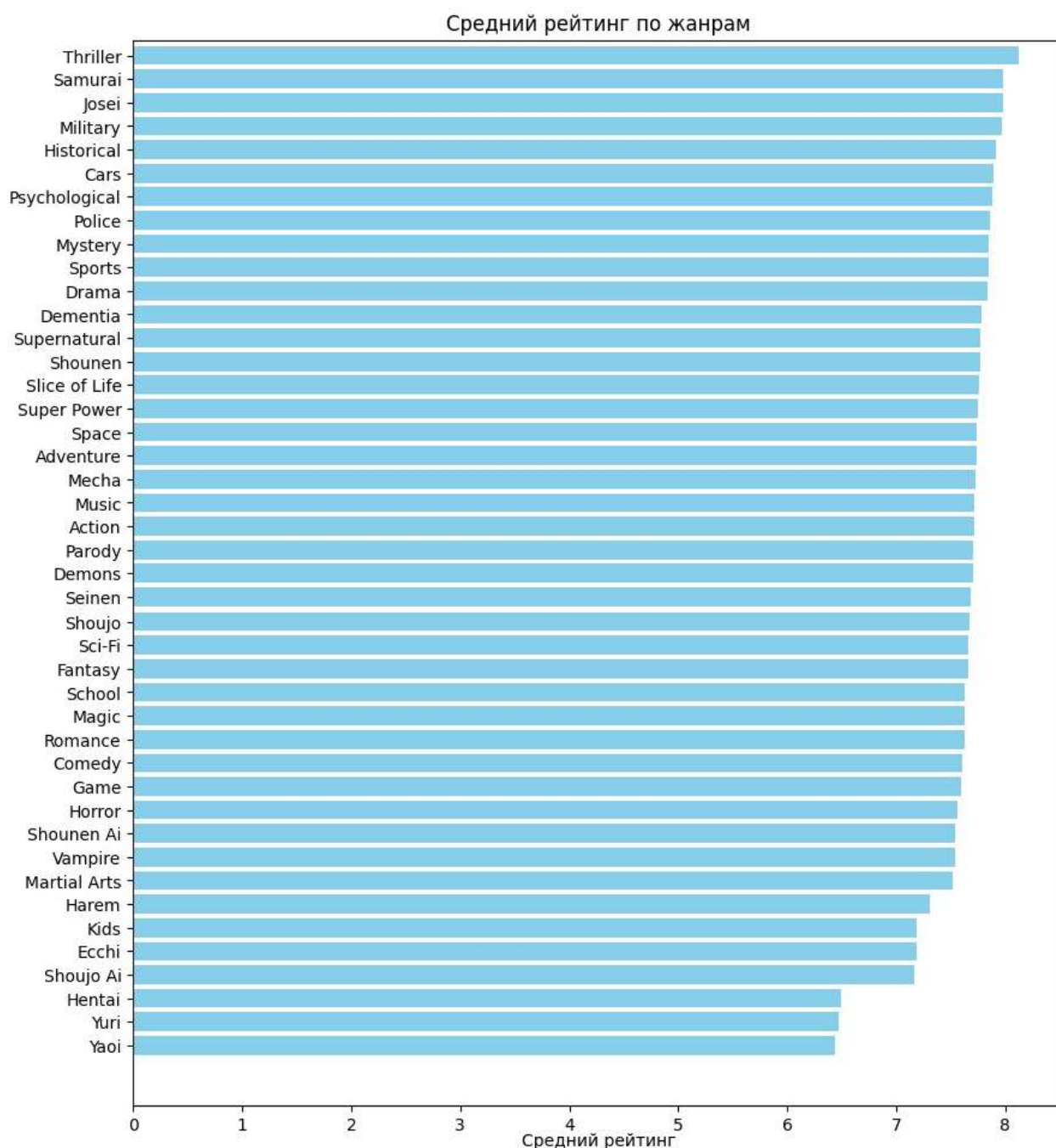
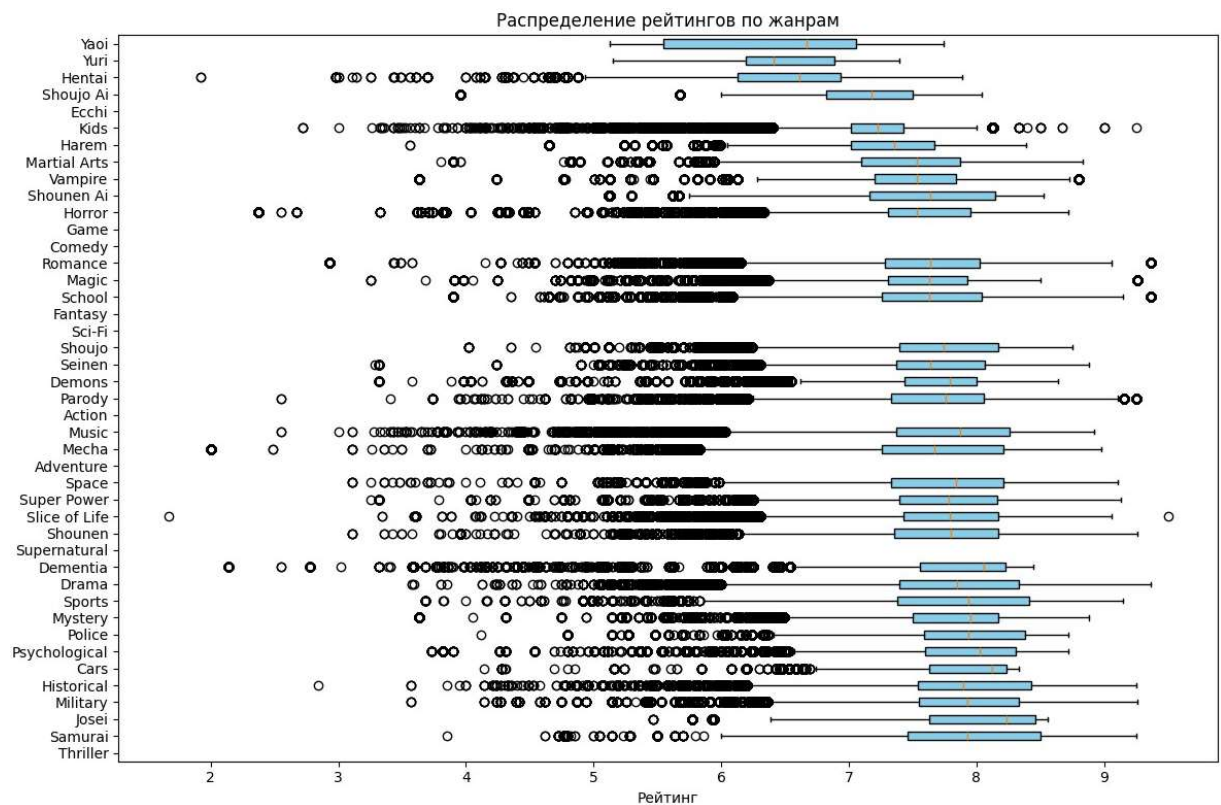


Рисунок 8 – Диаграмма рейтинга аниме в зависимости от жанра

Как видно, график показывает более отчетливую картину постановки рейтинга в зависимости от жанра картины. Для более наглядного анализа были построены «ящики с усами» для каждого жанра (Рисунок 9).





Эмпирическим методом было выяснено, что число фанатов практически линейно зависит от числа эпизодов аниме (Рисунок 10).

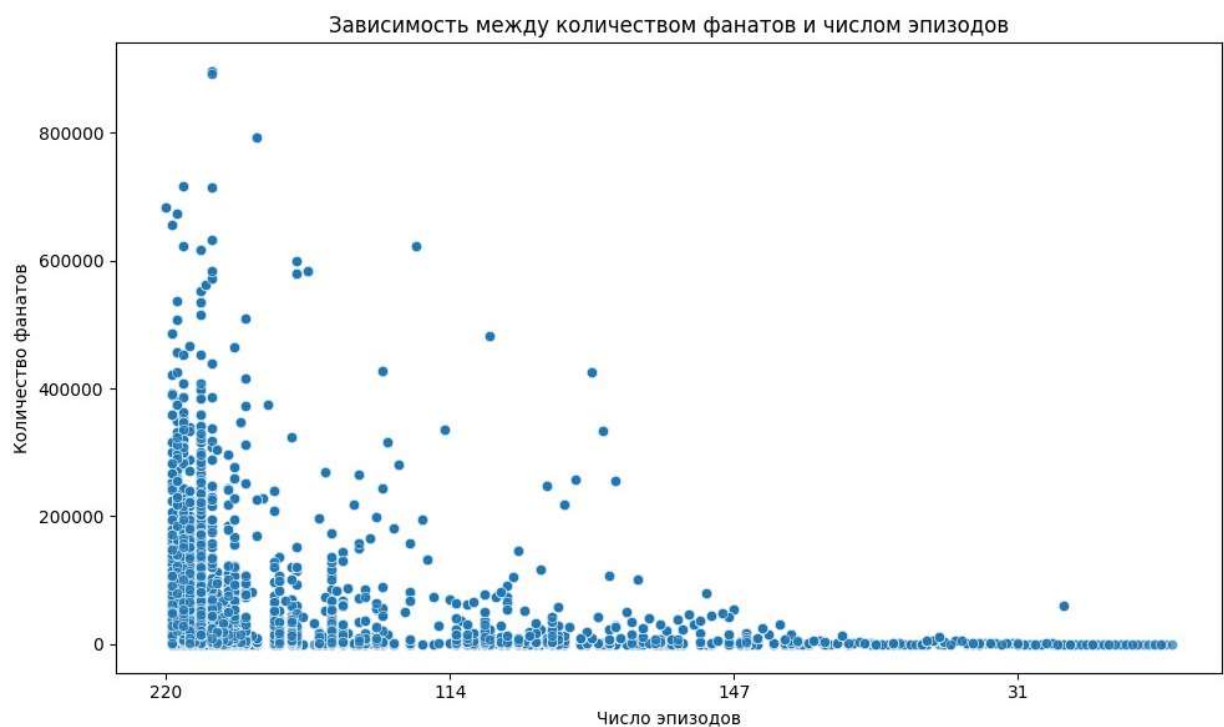


Рисунок 10 – Корреляция между количеством фанатов и эпизодов

В связи с этим фактом было принято решение при обучении всех моделей брать за основу лишь признак – «число эпизодов».



Было также принято решение проверить гипотезу: «Число слов в аниме влияет на общий рейтинг кинокартины» (Рисунок 11).

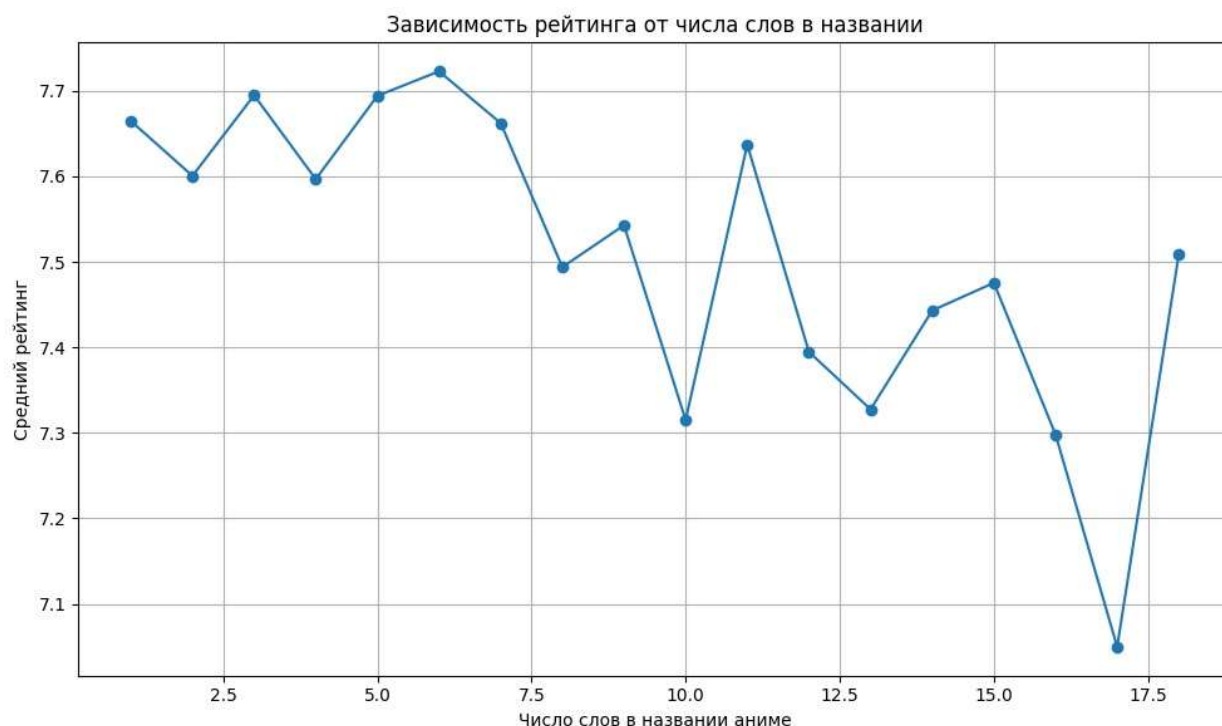


Рисунок 11 – Зависимость рейтинга от длины названия аниме

Данный график дает понимание, что чем увеличение длины названия кинокартины приводит к снижению рейтинга, проверим на boxplot-ax (Рисунок 12).

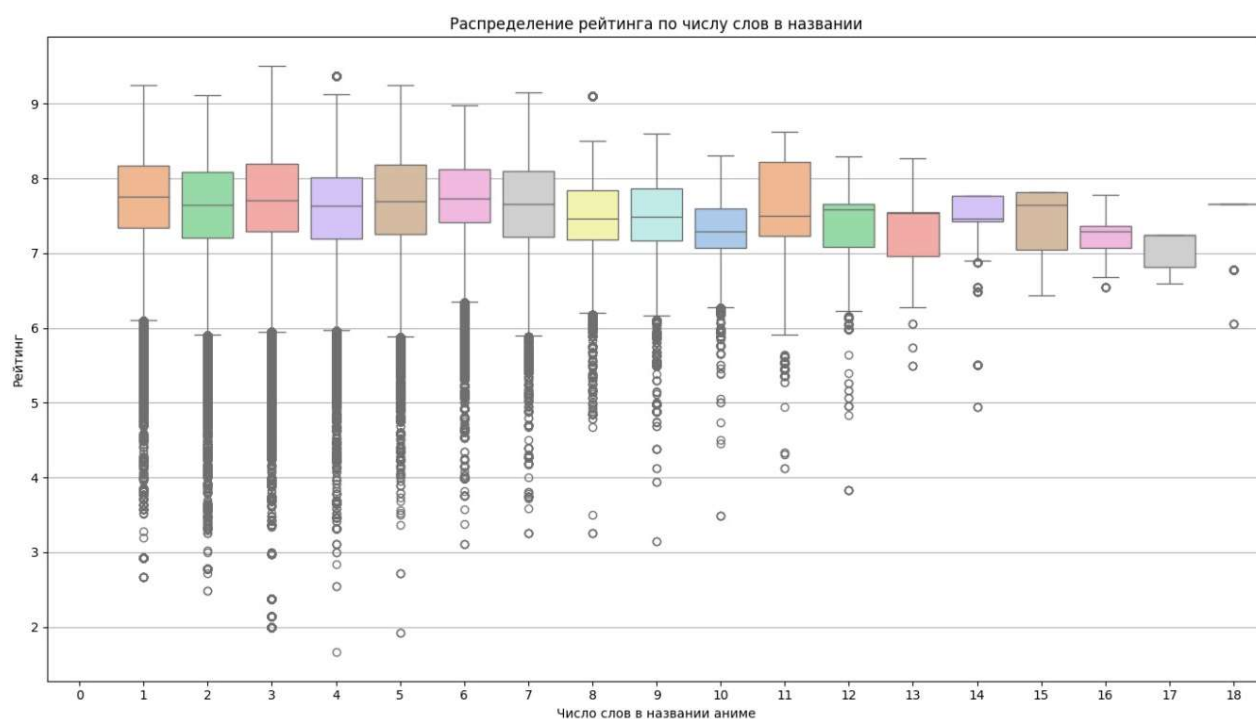


Рисунок 12 – «Ящики с усами» для рейтинга и длины названия



rating\_x - оценка отдельно взятого пользователя), поэтому с учетом анализа выше нельзя удалять какие-то низкоррелирующие признаки, поскольку они все таковыми являются.

Таким образом, итоговый датасет для обучения всех моделей представлен на рисунке 14.

	episodes	members	Action	Adventure	Drama	Ecchi	Harem	\
0	5.545410	2.875905	1.166207	-0.529897	-0.599222	-0.370334	-0.333033	
1	0.202223	0.038017	-0.857481	-0.529897	-0.599222	-0.370334	-0.333033	
2	0.147139	-0.073200	-0.857481	-0.529897	1.668831	2.700266	3.002703	
3	-0.155826	2.539763	1.166207	-0.529897	1.668831	-0.370334	-0.333033	
4	0.147139	0.704303	1.166207	-0.529897	1.668831	-0.370334	-0.333033	

	Hentai	Historical	Josei	Kids	Military	Mystery	Psychological	
0	-0.113819	-0.22604	-0.10179	-0.131418	-0.250653	-0.353112	-0.260749	
1	-0.113819	-0.22604	-0.10179	-0.131418	-0.250653	-0.353112	-0.260749	
2	-0.113819	-0.22604	-0.10179	-0.131418	-0.250653	-0.353112	-0.260749	
3	-0.113819	-0.22604	-0.10179	-0.131418	-0.250653	-0.353112	3.835108	
4	-0.113819	-0.22604	-0.10179	-0.131418	-0.250653	-0.353112	-0.260749	

	Samurai	Shoujo Ai	Shounen	Slice of Life	Sports	Super Power	\
0	-0.12717	-0.096742	1.746715	-0.404092	-0.179414	2.936826	
1	-0.12717	-0.096742	1.746715	-0.404092	-0.179414	-0.340504	
2	-0.12717	-0.096742	-0.572503	-0.404092	-0.179414	-0.340504	
3	-0.12717	-0.096742	-0.572503	-0.404092	-0.179414	-0.340504	
4	-0.12717	-0.096742	-0.572503	-0.404092	-0.179414	-0.340504	

	Supernatural	Thriller	Yaoi	Yuri	type_Movie	type_Music	\
0	-0.571011	-0.193831	-0.061059	-0.032008	-0.398299	-0.056446	
1	-0.571011	-0.193831	-0.061059	-0.032008	-0.398299	-0.056446	
2	-0.571011	-0.193831	-0.061059	-0.032008	-0.398299	-0.056446	
3	1.751279	-0.193831	-0.061059	-0.032008	-0.398299	-0.056446	
4	1.751279	-0.193831	-0.061059	-0.032008	-0.398299	-0.056446	

	type_ONA	type_OVA	type_Special	type_TV	rating_y
0	-0.109402	-0.341594	-0.278896	0.699568	7.81
1	-0.109402	-0.341594	-0.278896	0.699568	8.06
2	-0.109402	-0.341594	-0.278896	0.699568	7.31
3	-0.109402	-0.341594	-0.278896	0.699568	7.85
4	-0.109402	-0.341594	-0.278896	0.699568	7.74

Рисунок 14 – Итоговый датасет для обучения

*Пункт 3: линейная регрессия*

С помощью библиотеки «Sklearn» (для обучения модели регрессии Ridge) и «Optuna» (для нахождения оптимальной модели) был написан код, представленный на рисунке 15.

```
def objective(trial):
    alpha = trial.suggest_loguniform('alpha', 0.1, 100)
    model = Ridge(alpha=alpha)
    model.fit(X_scaled, y)
    y_pred = model.predict(X_scaled)
    mae = mean_absolute_error(y, y_pred)
    return mae

study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=50)

best_params = study.best_params
print("Лучшие гиперпараметры:", best_params)
best_model = Ridge(alpha=best_params['alpha'])
best_model.fit(X_scaled, y)
```

Рисунок 15 – Программный код для обучения модели линейной регрессии

В результате библиотеке для оптимизации удалось подобрать модель со следующими параметрами (Рисунок 16):

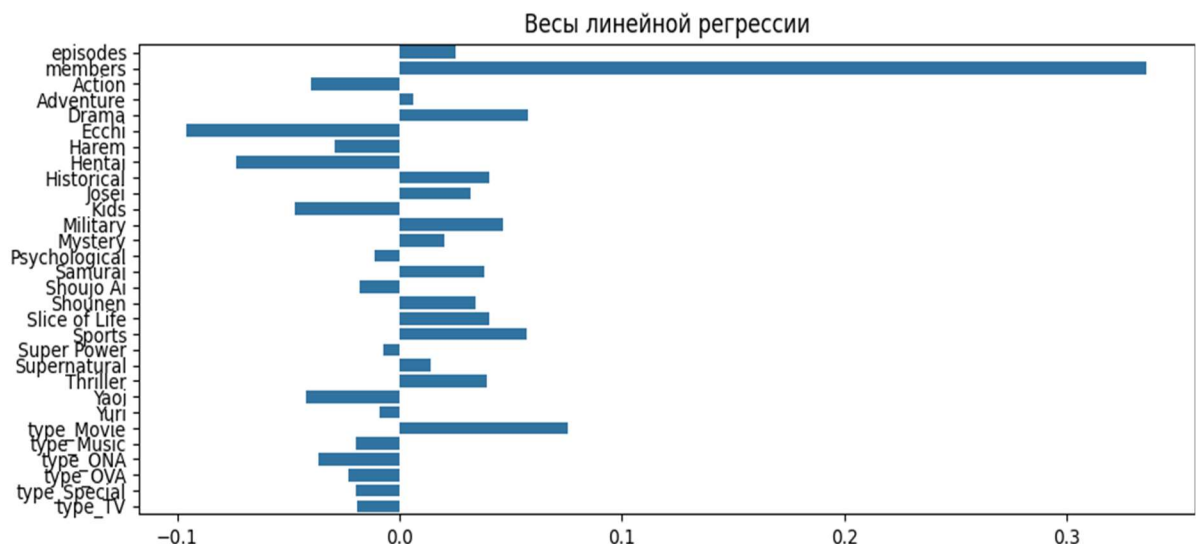


Рисунок 16 – Веса для модели линейной регрессии

В результате удалось получить следующие метрики ( $R^2$  – коэффициент детерминации, MAE – средняя абсолютная ошибка, MSE – средняя квадратическая ошибка, RMSE – среднеквадратическая ошибка):



MAE: 0.3862752392265896  
MSE: 0.2561124699226987  
RMSE: 0.5060755575234777  
 $R^2$ : 0.4327680748362943

Рисунок 17 – Метрики для линейной регрессии

Как видно, данная модель показывает себя с довольно плохой стороны, это можно увидеть на графике предсказаний и реальных значений (Рисунок 18).

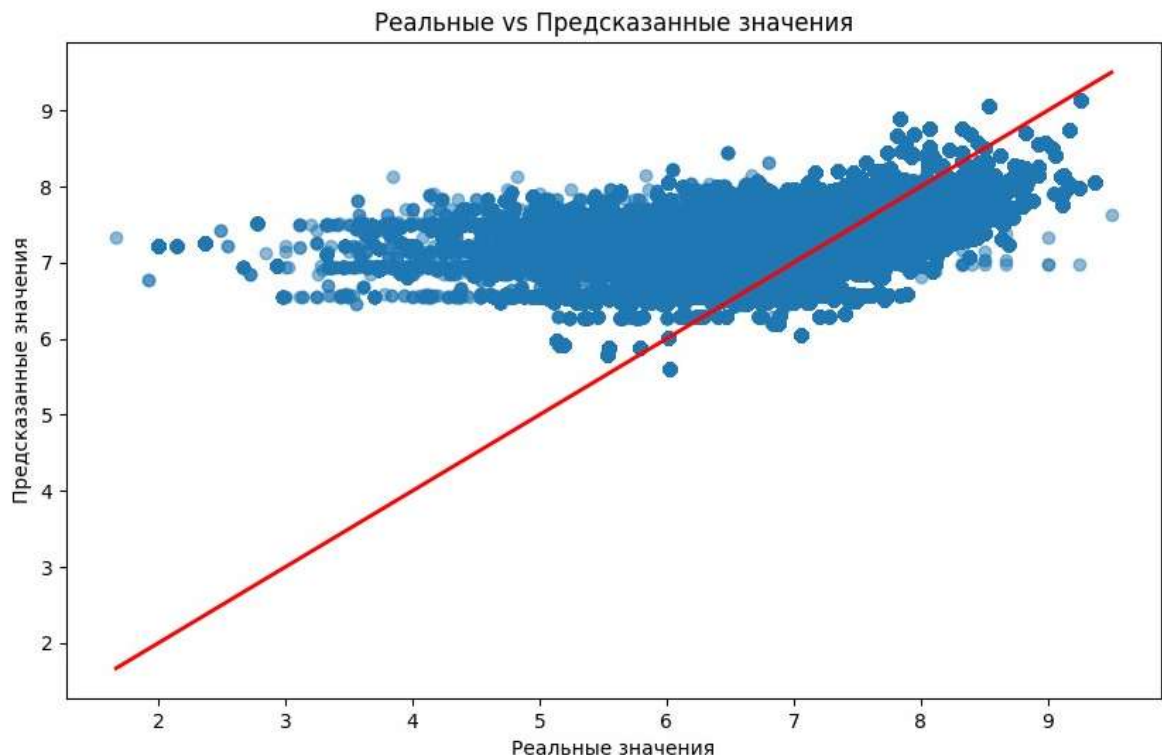


Рисунок 18 – Предсказания vs Реальные значения (линейная регрессия)

На аниме с высоким рейтингом прогнозирование можно считать довольно успешным, но уже средний и низкий рейтинг показывают довольно большие отклонения. Даже с использованием библиотеки «Optuna» не удалось получить хорошую модель.

#### *Пункт 4: дерево решений*

Модель для дерева решений была также взята из библиотеки «Sklearn» и также были выставлены гиперпараметры для оптимизации (глубина дерева, число лепестков на уровне, ответвлений, число признаков). Программный код и результаты оптимизации представлены на рисунках 19 и 20.

```
def objective(trial):
    params = {
        "max_depth": trial.suggest_int("max_depth", 3, 6),
        "min_samples_split": trial.suggest_int("min_samples_split", 2, 8),
        "min_samples_leaf": trial.suggest_int("min_samples_leaf", 1, 3),
        "max_features": trial.suggest_categorical("max_features", ["sqrt", "log2", None])
    }
    model = DecisionTreeRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_absolute_error(y_test, y_pred) # минимизируем MAE


study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=50)
print("Лучшие параметры:", study.best_params)
best_model = DecisionTreeRegressor(**study.best_params, random_state=42)
best_model.fit(X_train, y_train)
```

Рисунок 19 – Программный код для обучения модели дерева решений

```
0.32809{'max_depth': 6, 'min_samples_split': 6, 'min_samples_leaf': 2, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 7, 'min_samples_leaf': 2, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 6, 'min_samples_leaf': 2, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 7, 'min_samples_leaf': 1, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 6, 'min_samples_leaf': 2, 'max_features': None}
0.34698{'max_depth': 5, 'min_samples_split': 6, 'min_samples_leaf': 2, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 2, 'min_samples_leaf': 1, 'max_features': None}
0.34698{'max_depth': 5, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None}
Лучшие {'max_depth': 6, 'min_samples_split': 3, 'min_samples_leaf': 3, 'max_features': None}
0.32809{'max_depth': 6, 'min_samples_split': 5, 'min_samples_leaf': 2, 'max_features': None}
```

Рисунок 20 – Результаты поиска оптимального дерева

Далее рассмотрим полученные метрики:

 Метрики дерева решений:

```
MAE: 0.3280900394204879
MSE: 0.19544443124715224
RMSE: 0.44209097621095167
R²: 0.5672909614418071
```

Рисунок 21 – Метрики для дерева решений

Заметно уменьшение ошибок и увеличения коэффициента детерминации, даже на графике (Рисунок 22) сравнения предсказаний и реальных значений видно «сужение» к идеальной прямой.

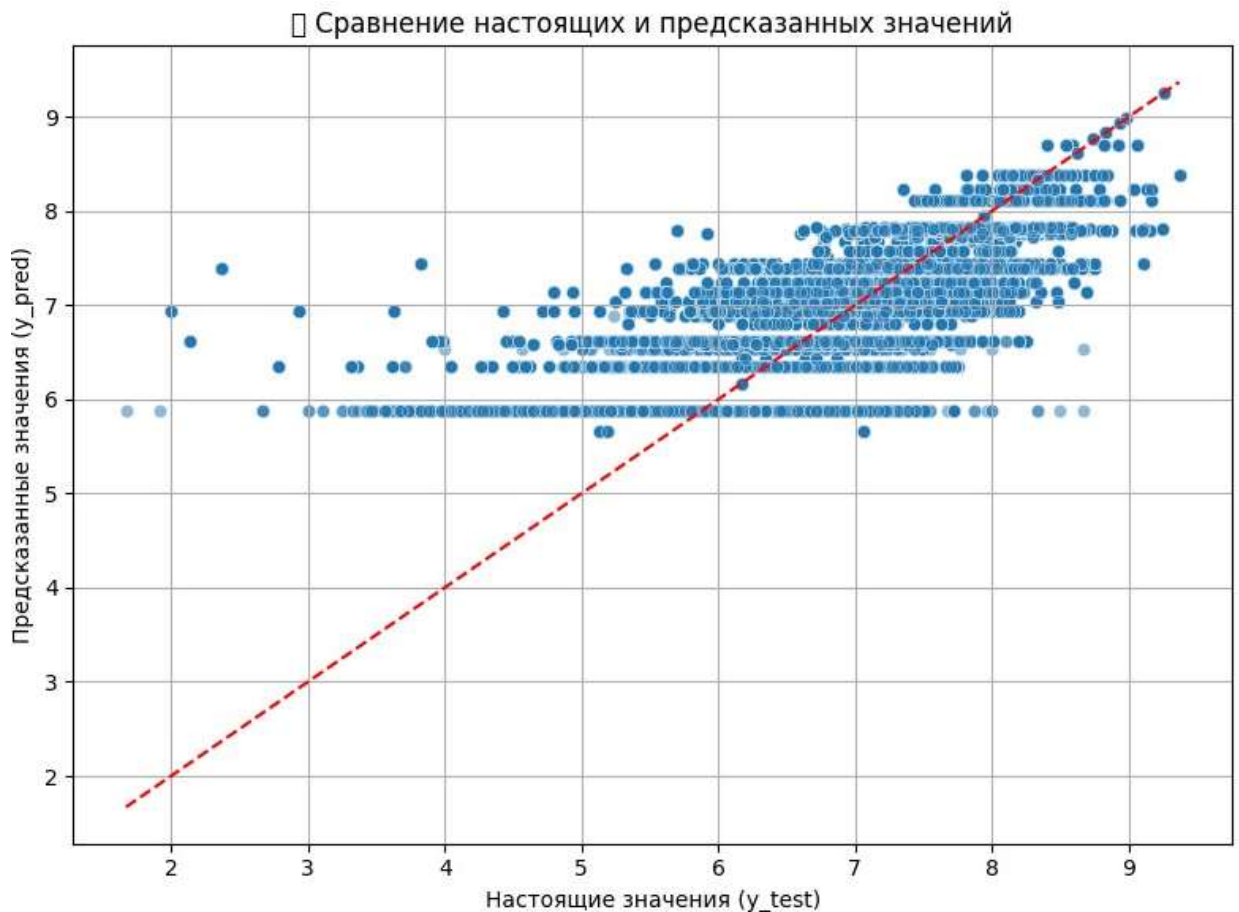


Рисунок 22 – Предсказания vs Реальные значения (дерево решений)

Если сравнить две гистограммы распределений остатков (насколько часто модель ошибается и на какую величину, рисунок 23), разница невелика.

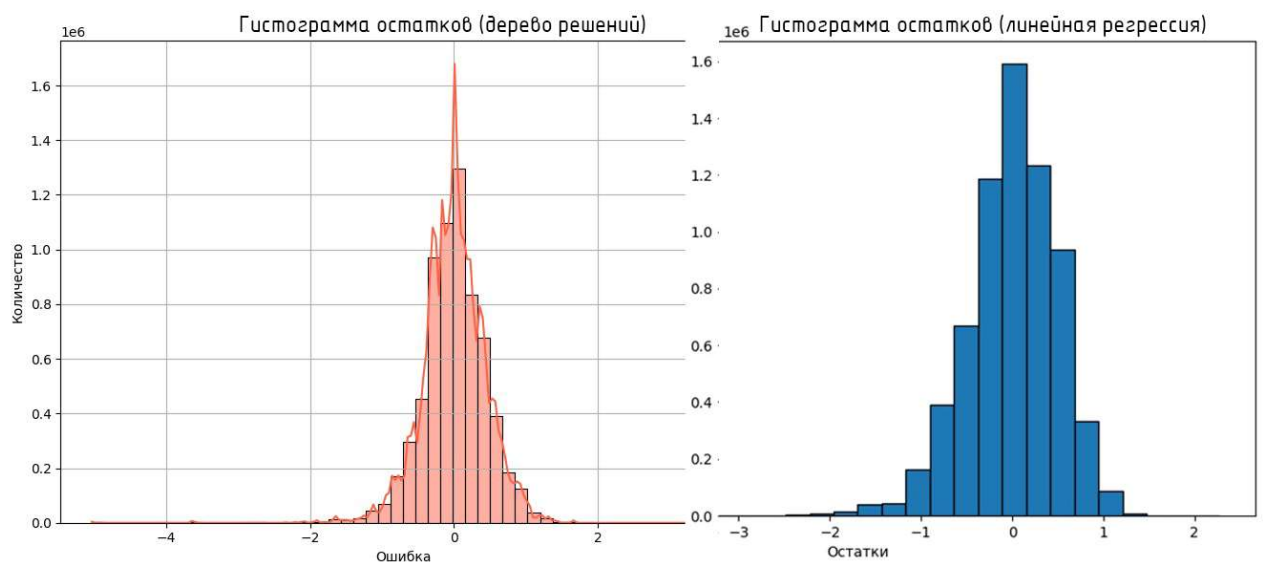


Рисунок 23 – Сравнение гистограмм остатков для двух моделей

На рисунке 24 представлена структура первых узлов дерева решений. Как видно, наибольший срез модель получает при опросе числа фанатов у аниме.



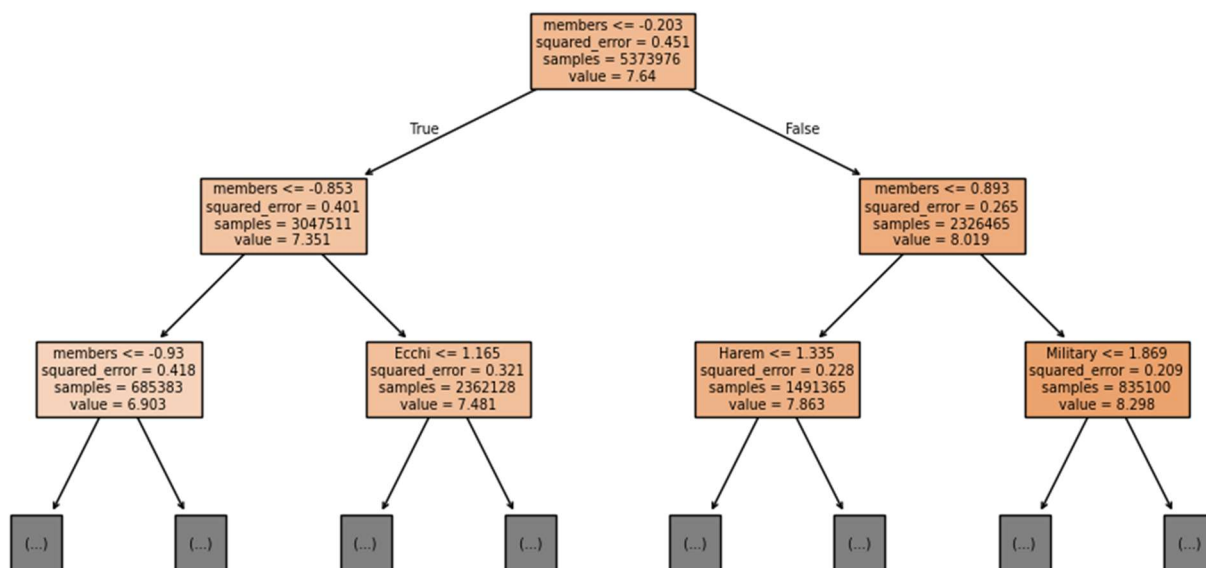


Рисунок 24 – Структура дерева решений (первые узлы)

В целом, линейный регрессор также имеет наибольший коэффициент в числе фанатов.

#### Пункт 5: CatBoost

Данная модель была разработана компанией «Яндекс» для работы с категориальными признаками, поэтому ее использование на уже закодированном датасете не совсем корректно, но поскольку все модели обучаются в относительно равных условиях, было принято решение не давать возможность модели использовать собственное кодирование категориальных признаков. На рисунке 25 представлен программный код для обучения и поиска оптимальной модели по тому же «таргету» (среднеквадратичная ошибка).


```
def objective(trial):
    params = {
        "iterations": trial.suggest_int("iterations", 100, 300),
        "depth": trial.suggest_int("depth", 3, 6),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.3, log=True),
        "l2_leaf_reg": trial.suggest_float("l2_leaf_reg", 1, 10),
        "bagging_temperature": trial.suggest_float("bagging_temperature", 0, 1),
        "random_strength": trial.suggest_float("random_strength", 0, 1),
        "verbose": 0
    }

    model = CatBoostRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_absolute_error(y_test, y_pred)

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=30)
print("Лучшие параметры:", study.best_params)
best_model = CatBoostRegressor(**study.best_params, random_state=42)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
```

Рисунок 25 – Программный код для обучения модели CatBoost

Полученные в результате обучения метрики представлены на рисунке 26.

 Метрики CatBoost:

MAE: 0.1865231654783405  
MSE: 0.07974499165211399  
RMSE: 0.28239155733150734  
R<sup>2</sup>: 0.8234466009216614

Рисунок 26 – Метрики для CatBoost

График предсказаний и использованных весов (признаков) представлены на рисунках 27 и 28 соответственно.

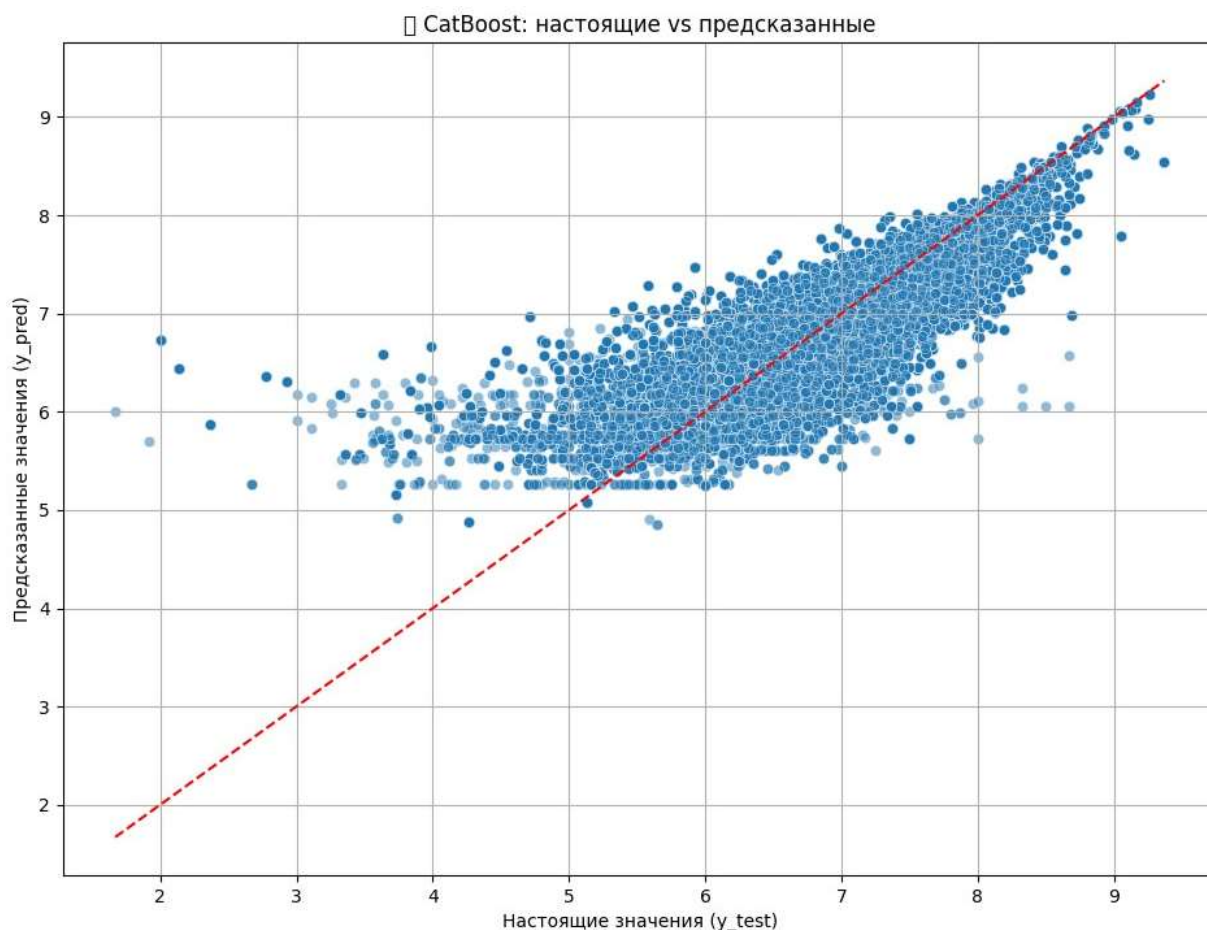


Рисунок 27 – Предсказания vs Реальные значения (CatBoost)

Видно, насколько сильно (по сравнению с деревом решений) уменьшилась дисперсия при оценке «проходных» аниме (кинокартин со средним рейтингом).

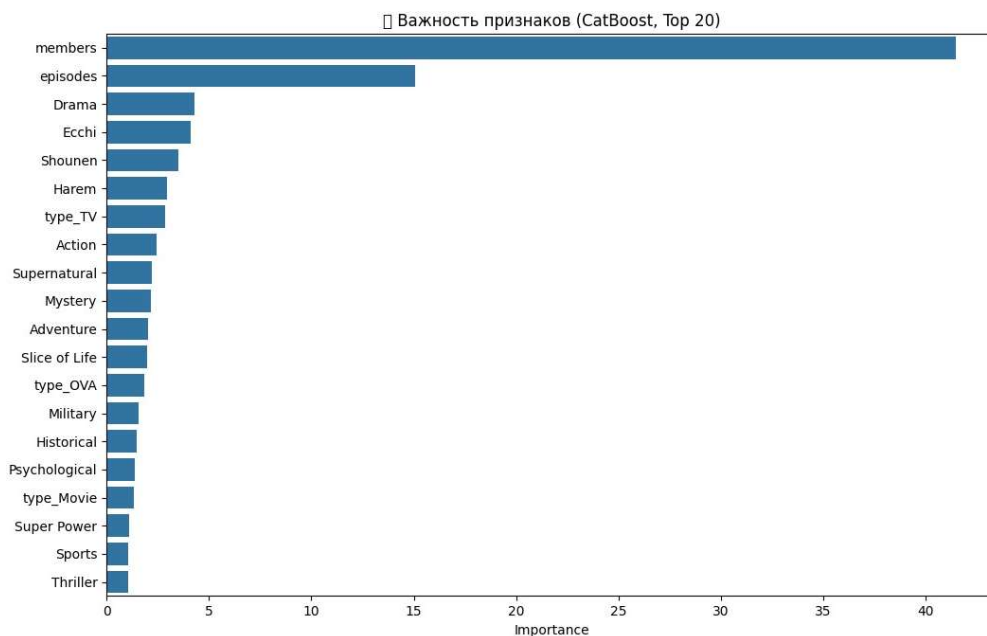


Рисунок 28 – Коэффициенты для признаков (CatBoost)

Ансамбль деревьев также посчитал, что число фанатов у картины является наиболее значимым признаком для предсказания оценки. Так что можно сделать вывод уже на данном этапе, что для любой новой кинокартины модели работать уже не будут.

### *Пункт 6: XGBoost*

Данная модель также представляет из себя модель градиентного бустинга, в структуре которой тоже лежит ансамбль деревьев. Однако эта модель более «универсальная», поскольку не специализирована на чисто категориальных признаках, как CatBoost.

```
def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 100, 300),
        "max_depth": trial.suggest_int("max_depth", 3, 6),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.3, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 5),
        "reg_alpha": trial.suggest_float("reg_alpha", 0, 5),
        "reg_lambda": trial.suggest_float("reg_lambda", 0, 5),
        "random_state": 42,
        "verbosity": 0
    }
    model = XGBRegressor(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_absolute_error(y_test, y_pred)

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=30)
print("Лучшие параметры:", study.best_params)
best_model = XGBRegressor(**study.best_params)
best_model.fit(X_train, y_train)
y_pred = best_model.predict(X_test)
```

Рисунок 28 а) – Программный код для обучения модели XGBoost

```
0.1940: {'n_estimators': 125, 'max_depth': 6, 'learning_rate': 0.10608829458349683, 'subsample': 0.836
0.3558: {'n_estimators': 162, 'max_depth': 6, 'learning_rate': 0.006640464333402818, 'subsample': 0.78
0.1815: {'n_estimators': 100, 'max_depth': 5, 'learning_rate': 0.2537161519639011, 'subsample': 0.88617
0.1536: {'n_estimators': 125, 'max_depth': 6, 'learning_rate': 0.17431347290802776, 'subsample': 0.856
0.1569: {'n_estimators': 141, 'max_depth': 6, 'learning_rate': 0.1527560252552215, 'subsample': 0.83933
0.1754: {'n_estimators': 139, 'max_depth': 6, 'learning_rate': 0.15658621141482726, 'subsample': 0.846
0.2025: {'n_estimators': 220, 'max_depth': 6, 'learning_rate': 0.0587755965097152, 'subsample': 0.79204
0.2622: {'n_estimators': 148, 'max_depth': 6, 'learning_rate': 0.025125810601175233, 'subsample': 0.85
0.2031: {'n_estimators': 127, 'max_depth': 6, 'learning_rate': 0.08986696838167592, 'subsample': 0.945
0.1565: {'n_estimators': 169, 'max_depth': 6, 'learning_rate': 0.16570648154980583, 'subsample': 0.655
0.2275: {'n_estimators': 173, 'max_depth': 6, 'learning_rate': 0.04392574171304275, 'subsample': 0.6065
{'n_estimators': 131, 'max_depth': 6, 'learning_rate': 0.28298525357832766, 'subsample': 0.8478747565
```

Рисунок 28 б) – Результат оптимизации модели XGBoost

Полученные в результате обучения метрики представлены на рисунке 29.

Метрики XGBoost:  
MAE: 0.13537644618837705  
MSE: 0.05163918666446847  
RMSE: 0.22724257229768471  
 $R^2$ : 0.8856721438880353

Рисунок 29 – Метрики для XGBoost

График предсказаний и использованных весов (признаков) представлены на рисунках 30 и 31 соответственно.

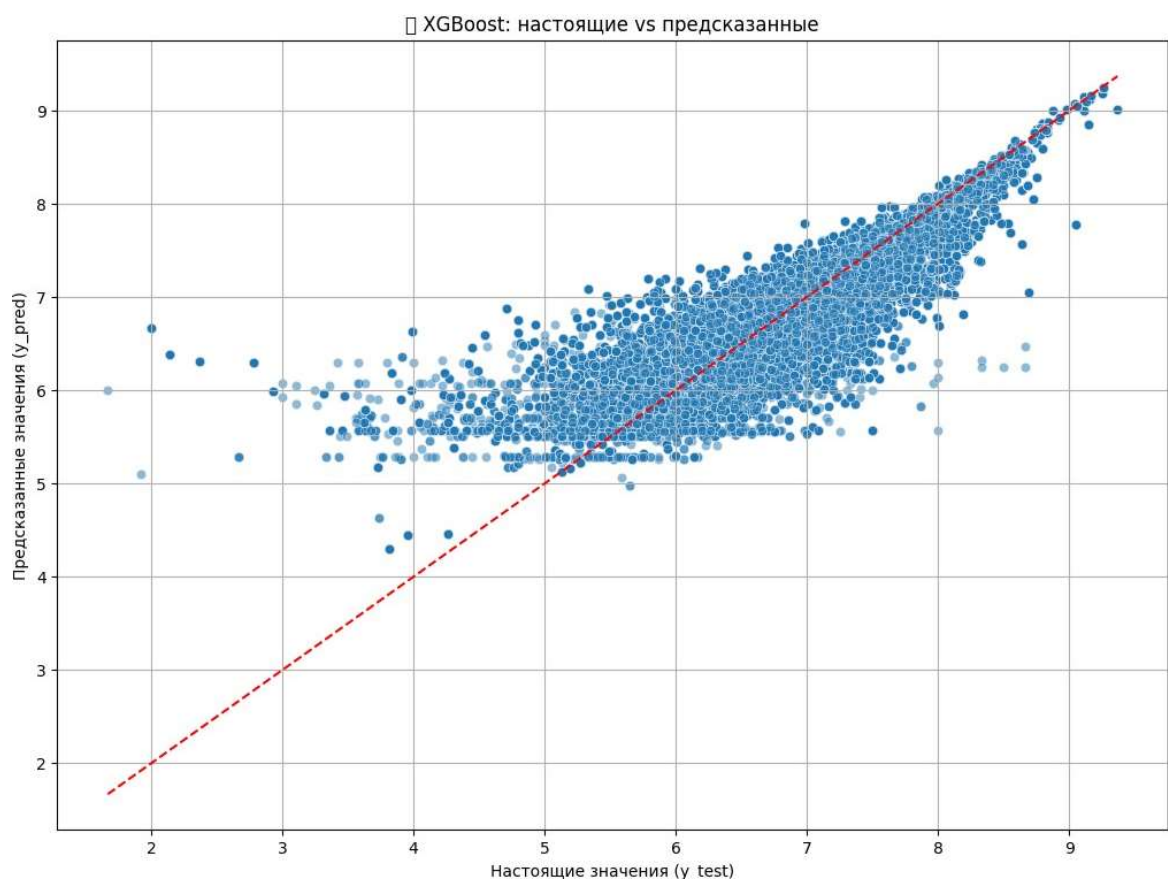


Рисунок 30 – Предсказания vs Реальные значения (XGBoost)



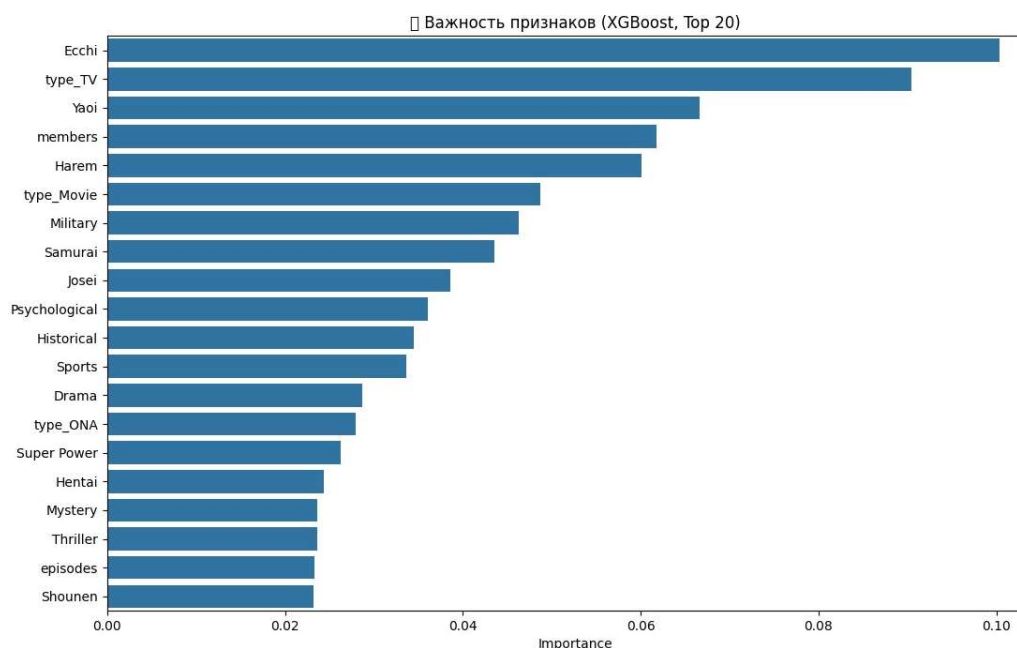


Рисунок 31 – Коэффициенты для признаков (XGBoost)

Эта модель не берет число участников и эпизодов как очень важный признак для прогноза, поэтому в целом ее использование может быть полезно для оценки новых аниме.

### Пункт 7: нейронная сеть

Алгоритм обучения тот же, заданные гиперпараметры: число нейронов на каждом слое, функции активации, число слоев.

```
def objective(trial):
    n_layers = trial.suggest_int("n_layers", 2, 4)
    hidden_layer_sizes = tuple(
        trial.suggest_int(f"n_units_l{i}", 4, 32, step=8) for i in range(n_layers)
    )
    params = {
        "hidden_layer_sizes": hidden_layer_sizes,
        "activation": trial.suggest_categorical("activation", ["relu"]),
        "solver": "adam",
        "alpha": trial.suggest_float("alpha", 1e-4, 1e-2, log=True),
        "learning_rate": trial.suggest_categorical("learning_rate", ["constant", "adaptive"]),
        "early_stopping": True,
        "max_iter": 100,
        "random_state": 42
    }
    model = MLPRegressor(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_absolute_error(y_test, y_pred)

study = optuna.create_study(direction="minimize")
study.optimize(objective, n_trials=10)
best_params = study.best_params.copy()
n_layers = best_params.pop("n_layers")
hidden_layer_sizes = tuple(best_params.pop(f"n_units_l{i}") for i in range(n_layers))
best_params["hidden_layer_sizes"] = hidden_layer_sizes
```

Рисунок 32 – Программный код для обучения модели нейронной сети


```

0.2226 {'n_layers': 1, 'n_units_l0': 32, 'activation': 'relu', 'alpha': 0.0024, 'learning_rate': 'adaptive'}.
0.1163 {'n_layers': 2, 'n_units_l0': 64, 'n_units_l1': 64, 'activation': 'relu', 'alpha': 0.00018, 'learning_rate': 'adaptive'}.
0.2279 {'n_layers': 1, 'n_units_l0': 32, 'activation': 'relu', 'alpha': 0.00038, 'learning_rate': 'constant'}.
0.2178 {'n_layers': 1, 'n_units_l0': 32, 'activation': 'relu', 'alpha': 0.0004, 'learning_rate': 'constant'}.
0.2235 {'n_layers': 1, 'n_units_l0': 32, 'activation': 'relu', 'alpha': 0.00042, 'learning_rate': 'adaptive'}.
0.2244 {'n_layers': 1, 'n_units_l0': 32, 'activation': 'relu', 'alpha': 0.00011, 'learning_rate': 'constant'}.
0.1797 {'n_layers': 1, 'n_units_l0': 64, 'activation': 'relu', 'alpha': 0.00011, 'learning_rate': 'adaptive'}.
Лучшие: {'n_layers': 2, 'n_units_l0': 64, 'n_units_l1': 64, 'activation': 'relu', 'alpha': 0.000182, 'learning_rate': 'constant'}

```

Рисунок 33 – Результат оптимизации модели нейросети

В результате наиболее оптимальной моделью с минимальной ошибкой является четырехслойная нейросеть с 64 нейронами в каждом слое.

 Метрики нейросети

```

MAE: 0.11631987679983212
MSE: 0.052116996563277194
RMSE: 0.22829147282208592
R²: 0.884551125078228

```

Рисунок 34 – Метрики для нейронной сети

График предсказаний и использованных весов представлены на рисунках 35 и 36 соответственно.

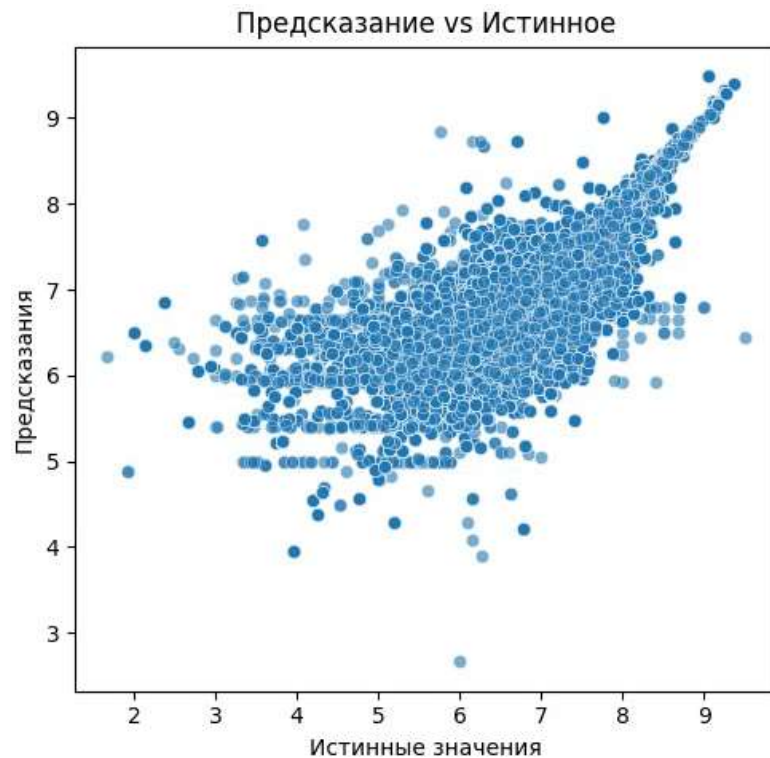


Рисунок 35 – Предсказания vs Реальные значения (Нейронная сеть)



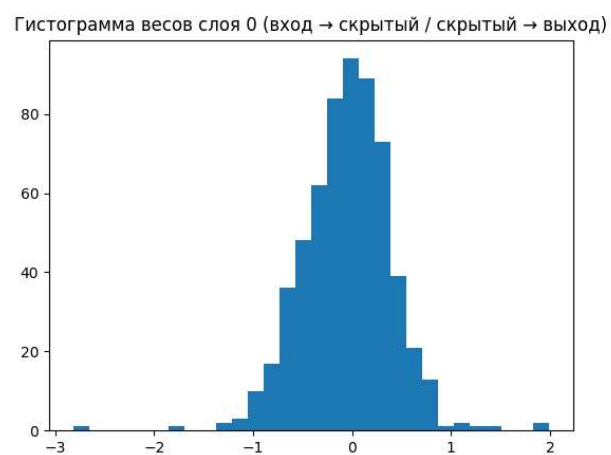
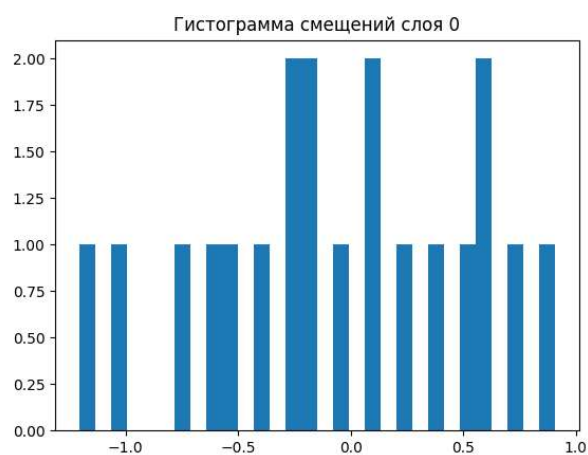


Рисунок 36 а) – Гистограммы смещений и весов первого слоя (входной)

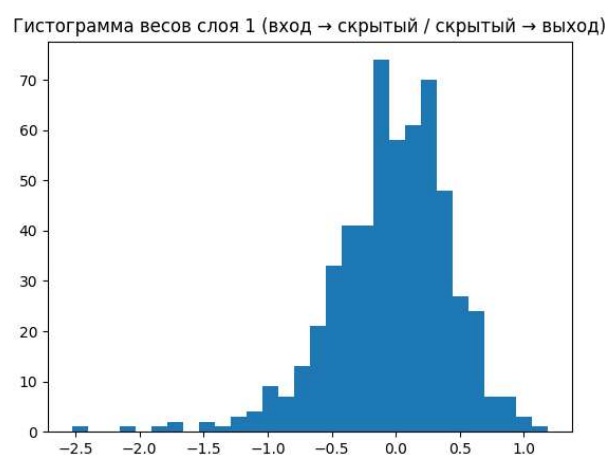
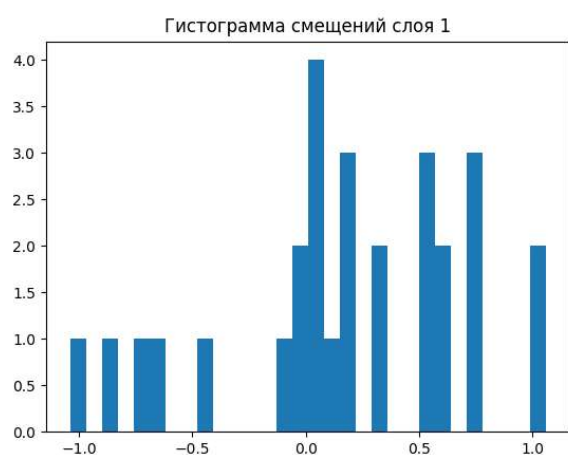


Рисунок 36 б) – Гистограммы смещений и весов второго слоя

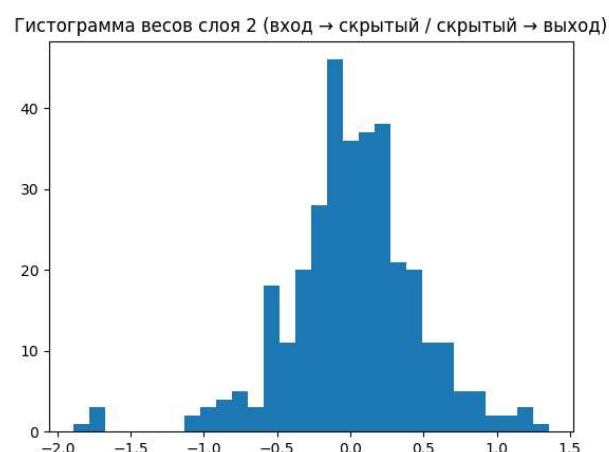
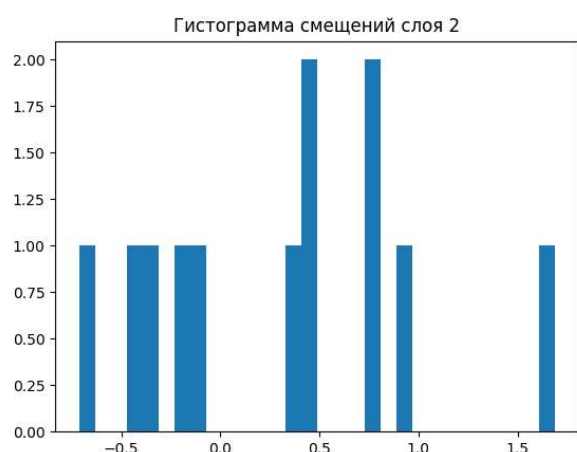


Рисунок 36 в) – Гистограммы смещений и весов третьего слоя

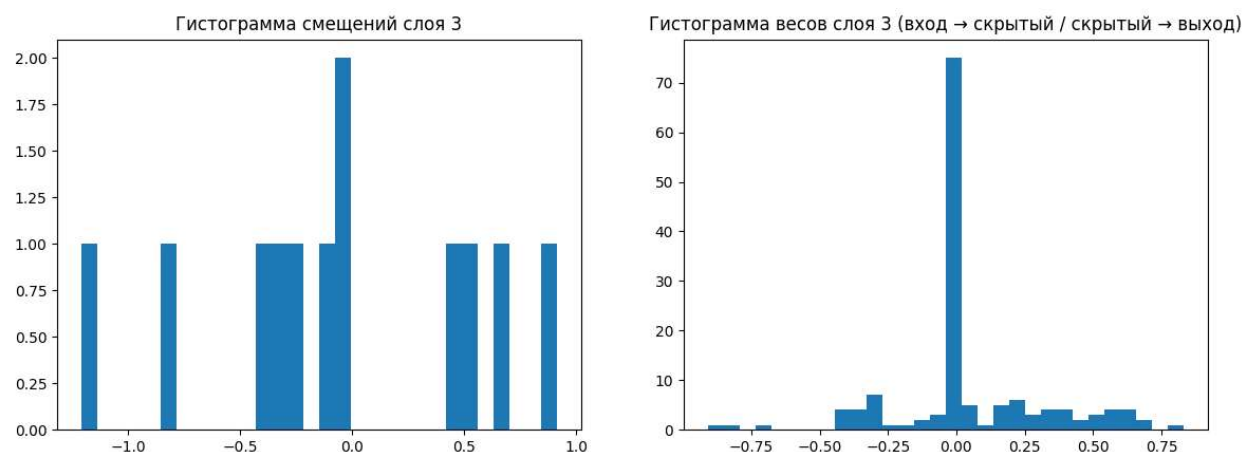


Рисунок 36 г) – Гистограммы смещений и весов четвертого слоя (выходной)

### *Пункт 8: сводная таблица*

В таблице 1 приведена сводная таблица с метриками разных моделей для сравнения.

Таблица 1 – Сравнительная таблица метрик моделей

Модель	MSE	MAE	R <sup>2</sup>
Линейная регрессия	0,256112	0,386275	0,432768
Дерево решений	0,195090	0,327935	0,567918
CatBoost	0,079506	0,186412	0,823911
XGBoost	0,051609	0,135407	0,885698
Нейросеть	0,081914	0,172191	0,818579

Как видно из таблицы, наилучшие метрики и точность прогнозирования показала модель XGBoost, которая использует ансамбль "слабых" деревьев, обучающихся последовательно. В отличие от линейной регрессии, XGBoost способен захватывать нелинейные зависимости между признаками, что важно для задач с категориальными признаками. CatBoost тоже показал отличные результаты, но немного уступил по метрикам. Это может быть связано с настройками по умолчанию или меньшей глубиной деревьев.


### *Дополнительный пункт: реальные условия*

Допустим, анонсируется новое аниме, у которого пока нет фанатов и количество эпизодов мало или неизвестно (как часто бывает). В этом случае из обучающего датасета удаляются столбцы «episodes» и «members».

В итоге смысла обучать и перепроверять модели линейного регрессора, дерева решений и нейросети нет. Поэтому зададим через «Optuna» гиперпараметры только для XGBoost и CatBoost.

```
def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 600),
        "max_depth": trial.suggest_int("max_depth", 5, 9),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.3, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 5),
        "reg_alpha": trial.suggest_float("reg_alpha", 0, 5),
        "reg_lambda": trial.suggest_float("reg_lambda", 0, 5),
        "random_state": 42,
        "verbosity": 0
    }
    model = XGBRegressor(**params)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    return mean_absolute_error(y_test, y_pred)
```

Рисунок 37 – Программный код для обучения модели XGBoost (доп.)

 Метрики XGBoost:

MAE:	0.33008875896331613
MSE:	0.21856235580650152
RMSE:	0.46750653022872474
R <sup>2</sup> :	0.5159753937105309

Рисунок 38 – Метрики для XGBoost (доп.)

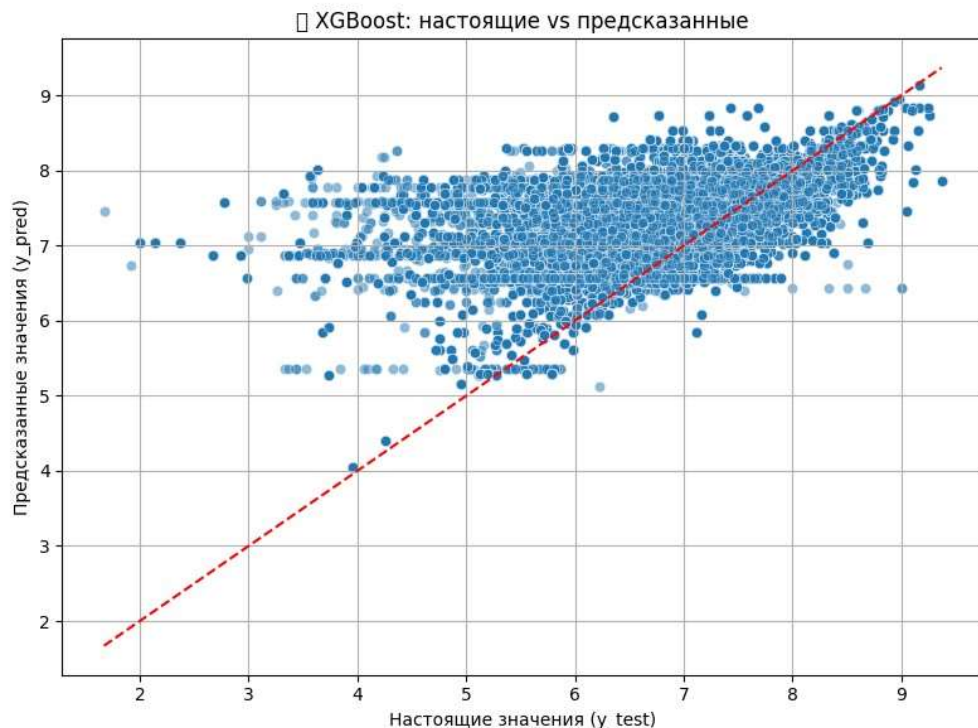


Рисунок 39 – Предсказания vs Реальные значения (XGBoost, доп.)

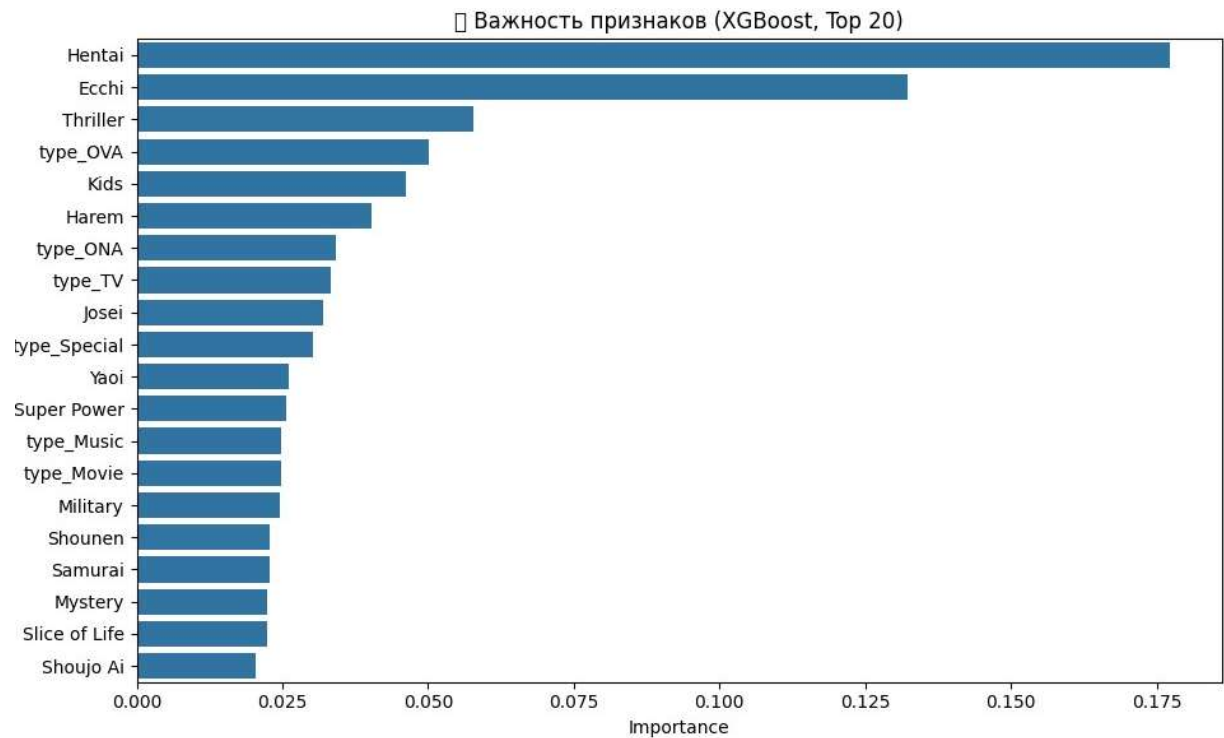


Рисунок 40 – Коэффициенты для признаков (XGBoost, доп.)

Как видно, модель значительно «упала» в качестве прогноза, что неудивительно. В реальных условиях предсказать рейтинг, зная лишь жанр и тип, практически невозможно, только если это не «Хентай».

```
def objective(trial):
    params = {
        "n_estimators": trial.suggest_int("n_estimators", 10, 600),
        "max_depth": trial.suggest_int("max_depth", 5, 9),
        "learning_rate": trial.suggest_float("learning_rate", 1e-3, 0.3, log=True),
        "subsample": trial.suggest_float("subsample", 0.5, 1.0),
        "colsample_bytree": trial.suggest_float("colsample_bytree", 0.5, 1.0),
        "gamma": trial.suggest_float("gamma", 0, 5),
        "reg_alpha": trial.suggest_float("reg_alpha", 0, 5),
        "reg_lambda": trial.suggest_float("reg_lambda", 0, 5),
        "random_state": 42,
        "verbosity": 0
    }
```

Рисунок 41 – Программный код для обучения модели CatBoost (доп.)

Метрики CatBoost:

MAE: 0.3145828785787108

MSE: 0.2096234073781972

RMSE: 0.4578464888783108

R<sup>2</sup>: 0.5357714422006121

Рисунок 42 – Метрики для CatBoost (доп.)

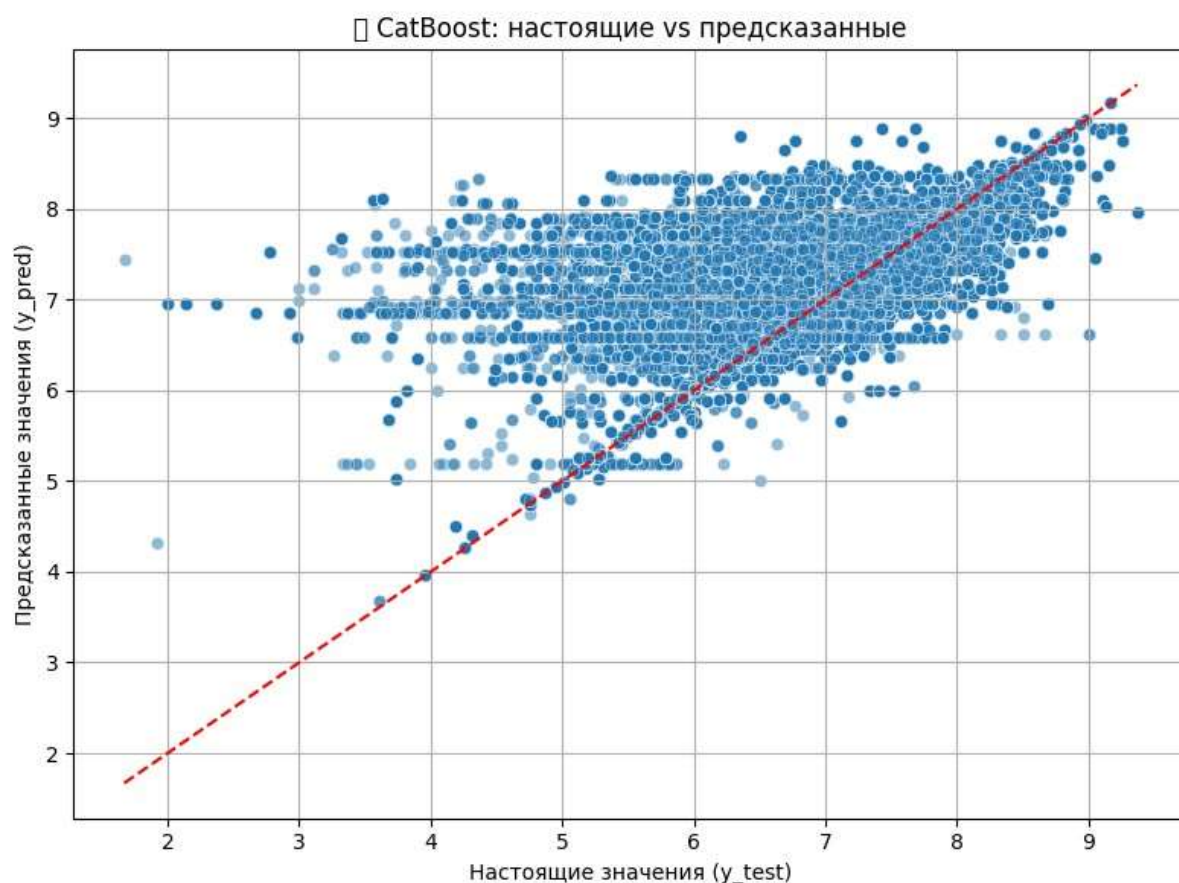


Рисунок 43 – Предсказания vs Реальные значения (CatBoost, доп.)

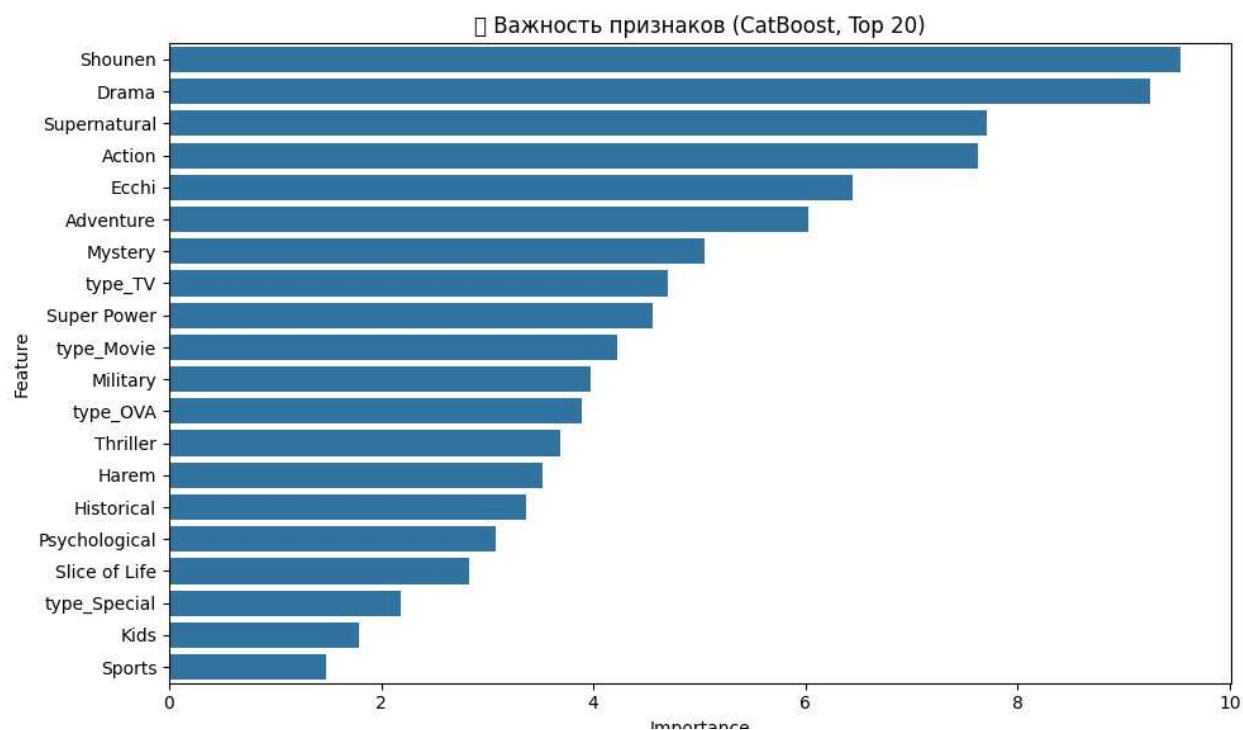


Рисунок 44 – Коэффициенты для признаков (CatBoost, доп.)

Сводка по метрикам в случае «усложненного» датасета представлена в таблице 2.

Таблица 2 – Сравнительная таблица метрик моделей (доп.)

Модель	MSE	MAE	$R^2$
CatBoost	0,209623	0,315828	0,53577
XGBoost	0,218562	0,330088	0,51597

Как видно, здесь уже победила модель от «Яндекса». В целом, модели практически одинаково плохо отработали, но следует понимать, что с наибольшей вероятностью линейный регрессор и дерево решений показали бы результат намного хуже, чем ансамбли деревьев.

**Вывод:** в результате лабораторной работы был произведен синтез моделей различных типов для задачи прогнозирования оценки аниме. Были рассмотрены следующие модели: линейная регрессия, дерево решений, CatBoost, XGBoost и нейросеть. В результате подготовки датасета (построения корреляционной карты, слияния признаков, создания дополнительных признаков, one-hot encoding и т.п.) и обучения всех моделей, были выведены необходимые графики и метрики для анализа эффективности каждой из моделей.

В результате наилучшим образом, как и следовало ожидать, показали себя модели градиентного бустинга (ансамбли деревьев) – CatBoost и XGBoost.