Открытый конкурс на лучшую работу студентов по естественным, техническим и гуманитарным наукам

Девиз: «Открой новые миры через анализ данных игр Epic Games Store»

Тема: Исследование набора данных о видеоиграх на платформе EGS.

Раздел: **07 Системный анализ, проектирование, управление и обработка информации**

Аннотация научной работы

1. Название: Исследование набора данных о видеоиграх на платформе EGS.

2. Код по [государственному рубрикатору научно-технической информации](http://www.grnti.ru/): 28.23.25

3. Вуз (название): 02069337 Ульяновский государственный технический университет

4. Год завершения работы 2023

5. Объем работы: 28 с.

6. Количество приложений: 10 с.

7. Количество иллюстраций: 16 ед.

8. Количество таблиц: 1 ед.

9. Количество источников литературы: 5 единиц

Характеристика работы:

1. Цель научной работы: см. главу «Введение»

2. Методы проведенных исследований: машинное обучение

3. Основные результаты научного исследования (научные, практические):

был проанализирован набор данных о видеоиграх на платформе EGS. Найдены наиболее значимые признаки, позволяющие определить успешность игр. Построены несколько моделей для предсказания, классификации и кластеризации видеоигр, опираясь на данные из площадки Epic Games Store.

4. Наличие документа об использовании научных результатов (да, нет): нет

Подпись автора (авторов) \_\_\_\_\_\_\_\_

СВЕДЕНИЯ

об авторе (авторах) и научном руководителе научной работы, представленной на открытый конкурс

|  |  |
| --- | --- |
| АВТОР (АВТОРЫ) | НАУЧНЫЙ РУКОВОДИТЕЛЬ |
| 1. Савенков | 1. Воронина |
| 2. Александр | 2. Валерия |
| 3. Владимирович | 3. Вадимовна |
| 4. Ульяновский государственный технический университет | 4. Ульяновский государственный технический университет |
| 5. ул. 1 Мая д. 25 | 5. 03 |
|  | 6. 2 |
|  | 7. 1 |
|  | 8. Домашний адрес: Ульяновск, Докучаева 18,56 |

Проректор по научной работе \_\_\_\_\_\_\_\_

Научный руководитель \_\_\_\_\_\_\_\_

Автор научной работы \_\_\_\_\_\_\_\_

Сведения о научной работе

1. Название работы: «Исследование набора данных о видеоиграх на платформе EGS»

2. Научный раздел конкурса, на который представляется работа (две цифры): 07

3. ГРНТИ (шесть символов): 28.23.25

4. Классификация работы: 3

5. Вид работы: теоретико-практическая

6. Возможность внедрения: нет

7. Возможность опубликования: да

8. Дополнительные признаки и сведения, если таковые имеются: нет

9. Ключевые слова (80 символов): Машинное обучения, нейронные сети, классификация, кластеризация, регрессия, Python

Подписи:

Автор (авторы) \_\_\_\_\_\_\_\_

Научный руководитель \_\_\_\_\_\_\_\_

Оглавление

[Введение 7](#_Toc153082629)

[Объект и цель исследования 8](#_Toc153082630)

[Данные анализа 9](#_Toc153082631)

[Методы анализа 11](#_Toc153082632)

[Инструменты 11](#_Toc153082633)

[Реализация и результаты 12](#_Toc153082634)

[Классификация 12](#_Toc153082635)

[Предсказание 17](#_Toc153082636)

[Кластеризация 23](#_Toc153082638)

[Получение результата 27](#_Toc153082639)

[Заключение 28](#_Toc153082640)

[Приложение 29](#_Toc153082641)

[Список литературы 39](#_Toc153082642)

# Введение

Введение: В современном цифровом эпосе развлечений видеоигры занимают особое место, привлекая внимание миллионов игроков по всему миру. С появлением платформы Epic Games Store, предоставившей уникальные возможности для приобретения и опыта игрового контента, интерес к этой форме развлечений стал еще более ярким и многообразным. В контексте активного взаимодействия геймеров с разнообразными играми и событиями, связанными с игровой индустрией, вопросы, касающиеся видеоигр в Epic Games Store, становятся ключевой темой обсуждения и исследования.

Согласно последним статистическим данным, количество игроков, предпочитающих игры на платформе Epic Games Store для приобретения, стремительно увеличивается. Несмотря на значительный интерес к этой теме, лишь немногие исследования посвящены подробному анализу и пониманию уникальных характеристик и особенностей этой игр на данной платформе в контексте игровой культуры.

Настоящая научная работа направлена на рассмотрение игр в Epic Games Store с фокусом на выявление ключевых особенностей в контексте цифровой игровой среды. Учитывая влияние, которое платформа может оказать на игровой опыт и предпочтения пользователей, а также в свете постоянно растущего интереса общества к технологиям и развлечениям, наше исследование стремится пролить свет на паттерны и тенденции, характерные для данного игрового окружения. В этом контексте, применение методов анализа данных и машинного обучения представляется перспективным подходом для выявления ключевых факторов, влияющих на восприятие и использование игр в Epic Games Store, и обеспечения более глубокого понимания этого аспекта игровой культуры.

# Объект и цель исследования

Исследовательский фокус направлен на набор статистических данных, который описывает атрибуты игр в Epic Games Store (EGS), включая 18 характеристик. Основной миссией нашего исследования является проведение анализа данных, связанных с этими характеристиками игр. Наша цель состоит в выявлении наиболее весомых и типичных факторов, влияющих на рейтинг и популярность игр.

Дополнительной целью является формирование наиболее точной модели, которая, опираясь на эти атрибуты, способна предсказывать уровень успешности игр. Этот подход открывает новые перспективы для более глубокого понимания динамики развития видеоигр и их взаимодействия с платформой EGS, что в свою очередь может способствовать более эффективному анализу и управлению в сфере игровой индустрии.

# Данные анализа

Информация для исследования была добыта из ресурса Kaggle, включая описания игр. Этот набор данных включает в себя 18 атрибутов и содержит около 20 тысяч записей. Файл в формате CSV доступен по указанной ссылке: [https://www.kaggle.com/datasets/mexwell/epic-games-store-dataset?select=games.csv](https://www.kaggle.com/datasets/mexwell/epic-games-store-dataset?select=games.csv%20) (Дата обращения: 08.09.23)

Данные содержат следующие признаки (таблица 1):

Таблица . Структура набора данных

|  |  |  |
| --- | --- | --- |
| Название признака | Описание | Пример значений |
| id | Идентификатор игры | 4c81547b81064acfb1902be7b06d6366 |
| name | Имя | Assassin's Creed® I: Director's Cut |
| game\_slug | Игровой сленг | assassins-creed-1 |
| price | Цена игры | 1999 |
| release\_date | Дата выпуска | 2008-04-09T15:00:00.000Z |
| platform | Платформа | Windows |
| description | Описание | You are an Assassin, a warrior shrouded in secrecy and feared for your ruthlessness. Your actions ca... |
| developer | Разработчик | Ubisoft |
| publisher | Издатель | Ubisoft |
| genres | Жанры | ACTION,RPG |
| id | ID | 62320d6a67855975e586e99b |
| company | Company | Tom's Guide |
| author | Author | Sherri L. Smith |
| rating | Rating | 90 |
| comment | Comment | Red Dead Redemption 2 serves up complex themes and characters, a heart-achingly beautiful world that... |
| date | Date | 2021-01-28T00:00:00.000Z |
| top\_critic | Top Critic | True |
| game\_id | Game ID | a3c78a5c62824677834c1008e0be9b2d |

# Методы анализа

## Инструменты

Используется Python 3.11 с PyCharm в качестве среды разработки и следующие библиотеки:

NumPy: Используется для эффективных вычислений с многомерными массивами, обеспечивая высокую производительность и удобство работы с числовыми данными.

Pandas: Применяется для хранения и анализа данных в табличной форме, предоставляя удобные структуры данных для манипуляций с информацией.

Matplotlib и Seaborn: Обе библиотеки используются для визуализации данных. Matplotlib предоставляет базовые инструменты для построения графиков, в то время как Seaborn улучшает визуальное оформление и облегчает создание сложных графиков.

Scikit-learn: Используется для применения разнообразных алгоритмов машинного обучения. Включает в себя модули для классификации, регрессии, кластеризации и других задач машинного обучения.

minisom: Эта библиотека предоставляет реализацию мини-пакета для обучения карт самоорганизации (SOM), что может быть полезно в кластерном анализе.

LogisticRegression: Этот метод машинного обучения из библиотеки Scikit-learn применяется для решения задачи логистической регрессии, часто используемой в бинарной классификации.

linkage и dendrogram: Они являются частями библиотеки SciPy и применяются для построения дендрограмм в задачах иерархической кластеризации.

MLPRegressor: Этот метод из библиотеки Scikit-learn предоставляет реализацию многослойного персептрона для решения задач регрессии, что может быть полезным в оценке линейных отношений в данных.

# Реализация и результаты

## Классификация

Давайте проведем классификацию для определения успешности игр, разделив их на два класса: успешные и неуспешные. Наша цель - построить модели классификации и оценить полученную точность, а также определить наиболее важные признаки игр для этой задачи. Это позволит нам получить краткое описание процесса классификации и его результатов:

Краткое описание работы классификации с помощью связей и дендрограммы:

Загрузка данных и их объединение:

* Импорт библиотек и загрузка данных из файлов.
* Объединение данных по общему ключу.

Подготовка данных для классификации:

* Создание целевой переменной, преобразование дат в Unix-время, и кодирование жанров в бинарные признаки.

Определение признаков и целевой переменной:

* Выделение признаков и целевой переменной.
* Разделение данных на обучающий и тестовый наборы.

Обучение модели дерева решений:

* Создание и обучение модели дерева решений с использованием Scikit-learn.

Оценка модели на тестовом наборе данных:

* Вычисление точности предсказаний.

Визуализация важности признаков:

* Вычисление и визуализация общей важности для категории "жанры", цены и даты релиза.

Интеграция с Flask:

* Создание веб-приложения с использованием Flask и Blueprint.
* Визуализация результатов и важности признаков.

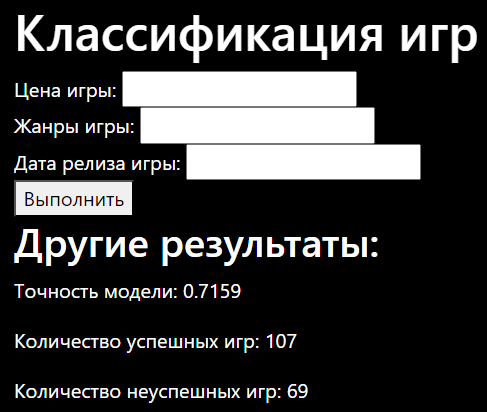


Рисунок 1. Результат выполнения классификации с помощью связей и дендрограммы

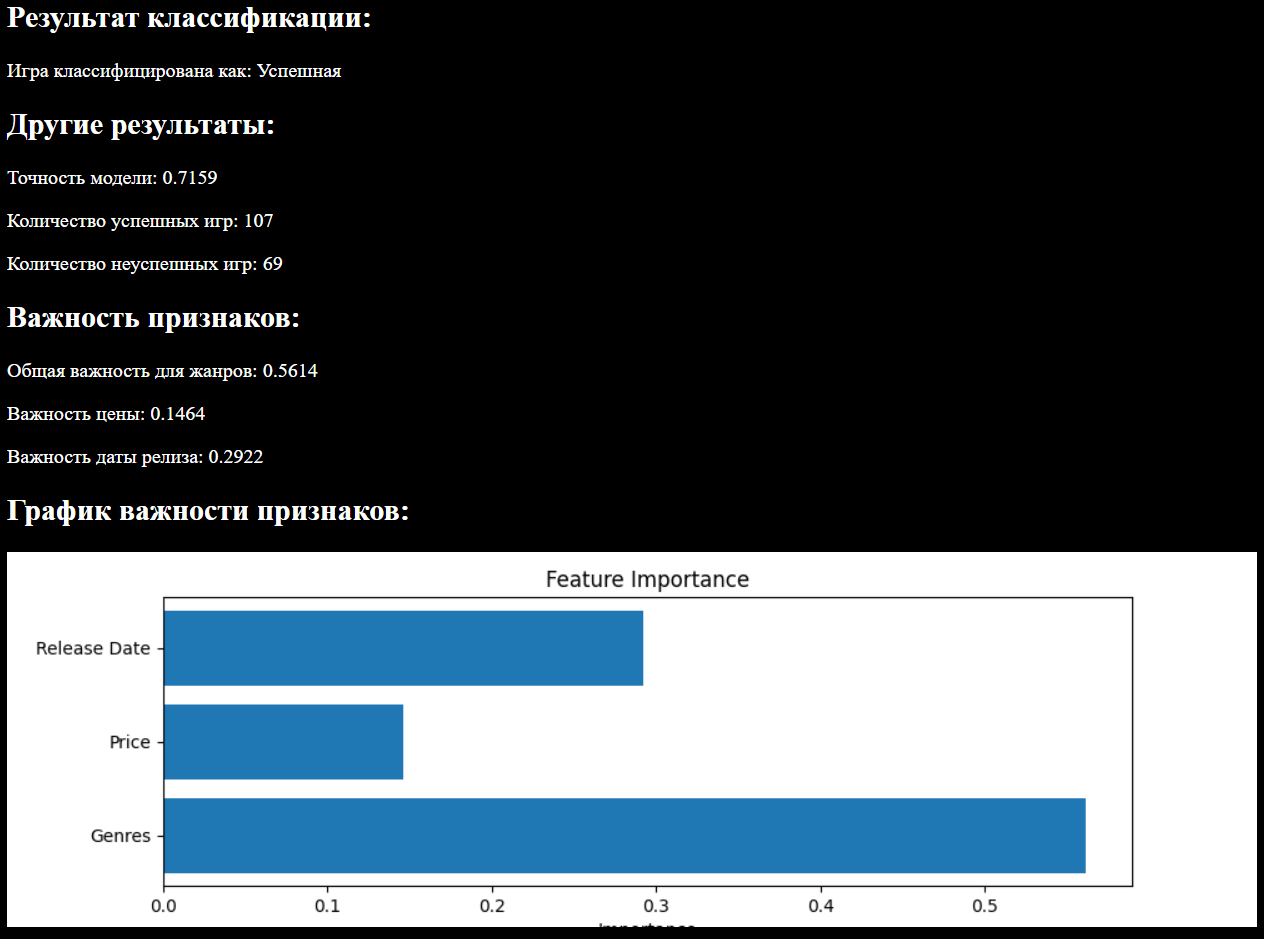


Рисунок 2. Результат выполнения классификации с помощью связей и дендрограммы

Краткое описание работы классификации с помощью логической регрессии:

Загрузка и предобработка данных:

* Загружаются данные из файлов "games.csv" и "open\_critic.csv".
* Данные объединяются и создается целевая переменная "Success" на основе рейтинга игр.

Подготовка данных для обучения:

* Преобразование столбца "release\_date" в формат Unix-времени.
* Преобразование жанров в бинарные признаки с использованием one-hot encoding.

Разделение данных на обучающий и тестовый наборы:

* Выделение признаков и целевой переменной.
* Разделение данных на обучающую и тестовую выборки.

Обучение модели логистической регрессии:

* Создание и обучение модели логистической регрессии с использованием библиотеки Scikit-learn.

Оценка модели на тестовом наборе данных:

* Предсказание успешности игр на тестовом наборе.
* Расчет точности модели.

Анализ результатов и визуализация:

* Получение матрицы ошибок, значений ROC-кривой и ее визуализация.
* Сохранение изображения ROC-кривой в байтовом массиве.

Интеграция с Flask:

* Создание веб-приложения с использованием Flask и Blueprint.
* Отображение результатов, матрицы ошибок, коэффициентов признаков и ROC-кривой в веб-интерфейсе.
* Возможность ввода пользователем параметров игры для предсказания ее успешности.

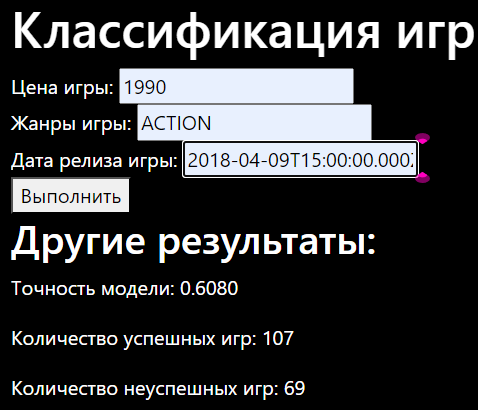


Рисунок 3. Результат выполнения классификации с помощью логической регрессии



Рисунок 4. Результат выполнения классификации с помощью логической регрессии

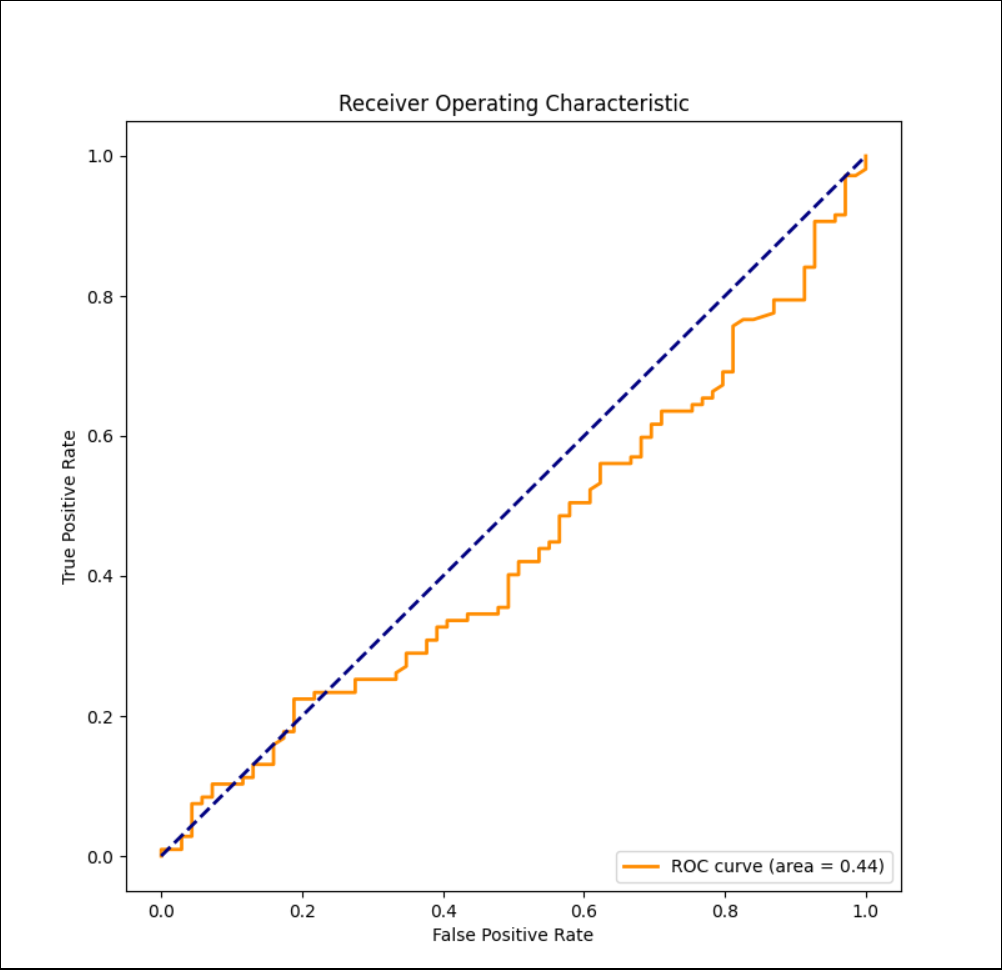


Рисунок 5. Результат выполнения классификации с помощью логической регрессии

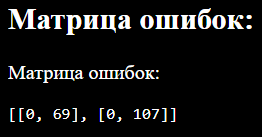


Рисунок 6. Результат выполнения классификации с помощью логической регрессии

Вывод: Сравнив и проанализировав результаты моделей: связь с дендрограммой и логистической регрессии на основе введенных и тестовых данных при выполнении классификации, пришел к выводу, что дерево решений более подходит на выполнения задачи классификации, так как точность модели 0.7159 и она выше, чем точности логической регрессии 0.608 и удобнее выводить важность признаков, но в свою очередь по логистической регрессии удобнее проверять точность самой модели.

## Предсказание

Решим задачу предсказания с помощью моделей линейной регрессии и нейронной сети MLPRegressor. Для предсказания рейтинга игры, опираясь на данных об ней. Краткое описание работы предсказания с помощью линейной регрессии:

Подготовка данных:

* Загрузка данных из двух CSV-файлов ("games.csv" и "open\_critic.csv") и их объединение.
* Преобразование даты релиза в числовой формат (в днях от минимальной даты).
* Преобразование категориального признака "жанры" в числовой формат с использованием LabelEncoder.
* Удаление записей с отсутствующим рейтингом.

Подготовка признаков и целевой переменной:

* Выделение признаков (жанр, цена, дата релиза) и целевой переменной (рейтинг).

Полиномиальное преобразование признаков и стандартизация:

* Применение полиномиального преобразования признаков с помощью PolynomialFeatures.
* Стандартизация числовых признаков с использованием StandardScaler.

Разделение данных на обучающий и тестовый наборы:

* Разделение данных на обучающий и тестовый наборы.

Обучение модели линейной регрессии:

* Создание и обучение модели линейной регрессии с использованием Scikit-learn.

Веб-интерфейс с Flask:

* Создание веб-приложения с использованием Flask и Blueprint.
* Отображение формы для ввода параметров игры и предсказания её рейтинга.
* Оценка точности модели на тестовом наборе данных.

Визуализация результатов:

* Создание и сохранение графика рассеяния фактических и предсказанных рейтингов.
* Создание и сохранение гистограммы остатков (разницы между фактическими и предсказанными значениями).

Отображение результатов в веб-интерфейсе:

* Отображение предсказанного рейтинга, точности предсказания и оценки модели в шаблоне HTML.

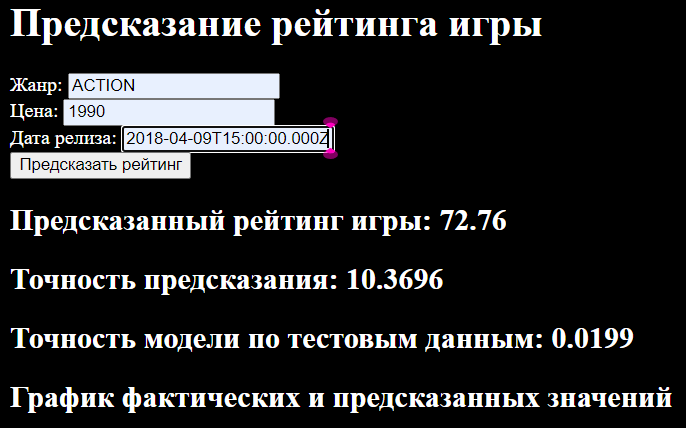


Рисунок 7. Результат выполнения предсказания с помощью линейной регрессии

## 

Рисунок 8. Результат выполнения предсказания с помощью линейной регрессии

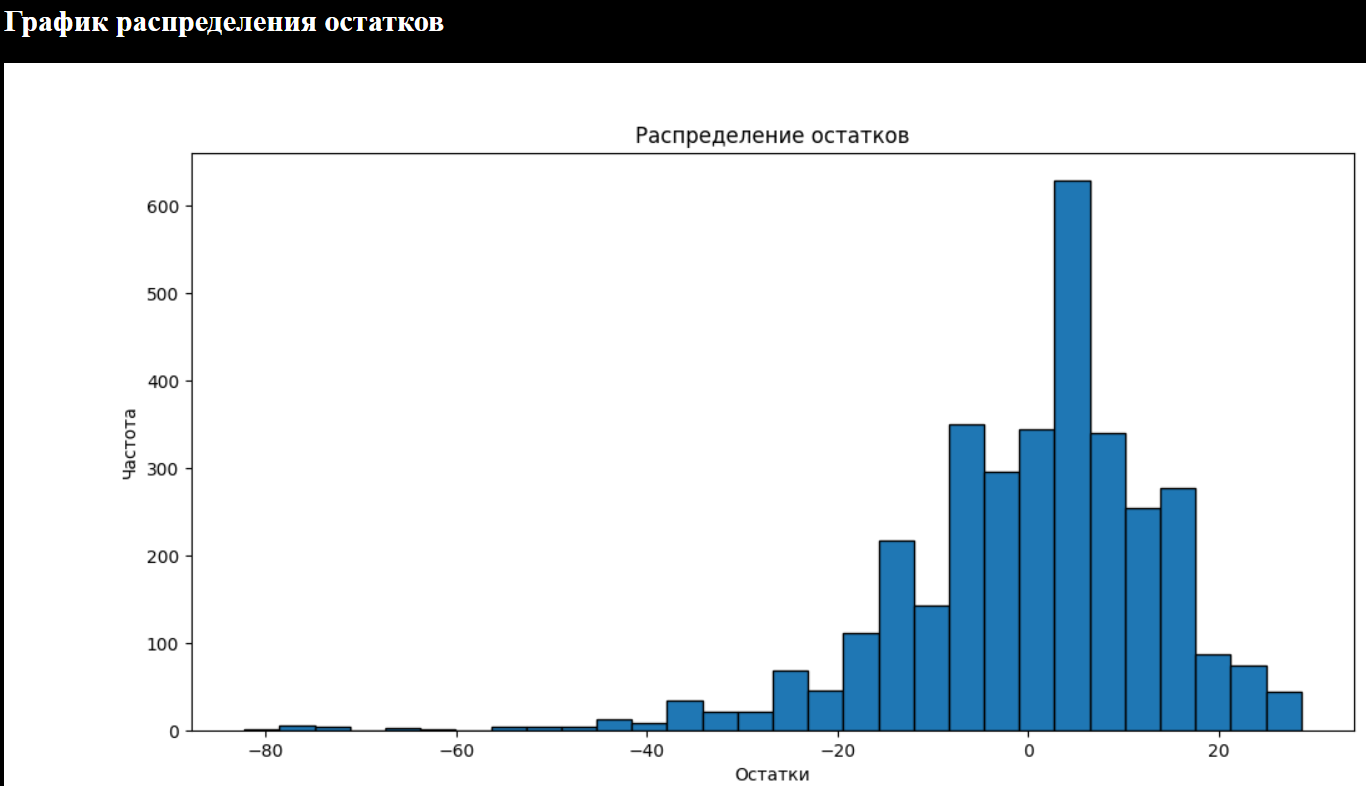


Рисунок 9. Результат выполнения предсказания с помощью линейной регрессии

Краткое описание работы предсказания с помощью нейронной сети MLPRegressor:

Подготовка данных:

* Загрузка данных из двух CSV-файлов ("games.csv" и "open\_critic.csv").
* Объединение данных по столбцам id и game\_id.
* Удаление строк с отсутствующими значениями в необходимых столбцах.

Выбор признаков и целевой переменной:

* Выбор признаков (жанры, цена, дата релиза) и целевой переменной (рейтинг).

Преобразование данных:

* Преобразование категориального признака "жанры" в числовой формат с использованием get\_dummies.
* Преобразование даты релиза в числовой формат (в данном случае, использование года релиза).

Разделение данных на обучающий и тестовый набор:

* Разделение данных на обучающий и тестовый наборы.

Нормализация данных:

* Применение MinMaxScaler для нормализации данных.

Инициализация и обучение MLPRegressor:

* Создание и обучение модели многослойного персептрона с использованием Scikit-learn.

Оценка точности модели:

* Расчет точности модели на тестовых данных.

Веб-интерфейс с Flask:

* Создание веб-приложения с использованием Flask и Blueprint.
* Отображение страницы с графиком рассеяния фактических и предсказанных рейтингов, точностью модели и стандартным отклонением.

Предсказание рейтинга на основе входных данных:

* Предсказание рейтинга на основе введенных пользователем данных.
* Оценка точности предсказания с использованием средней абсолютной ошибки (MAE) и среднеквадратичной ошибки (MSE).

Визуализация результатов в веб-интерфейсе

* Отображение предсказанного рейтинга, точности предсказания и MSE в шаблоне HTML.

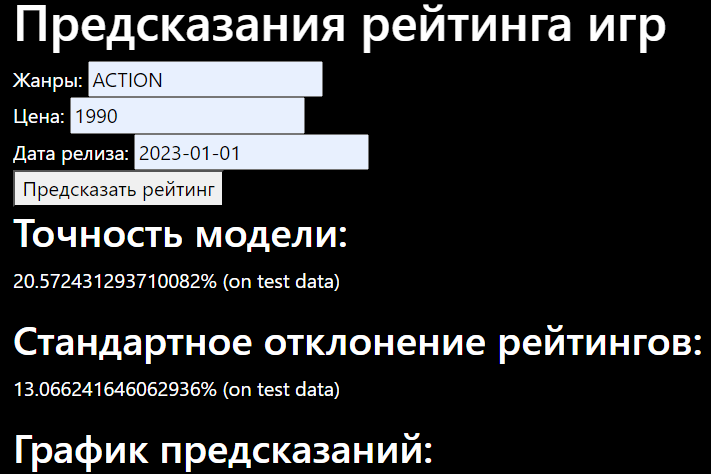


Рисунок 10. Результат выполнения предсказания с помощью нейронной сети MLPRegressor

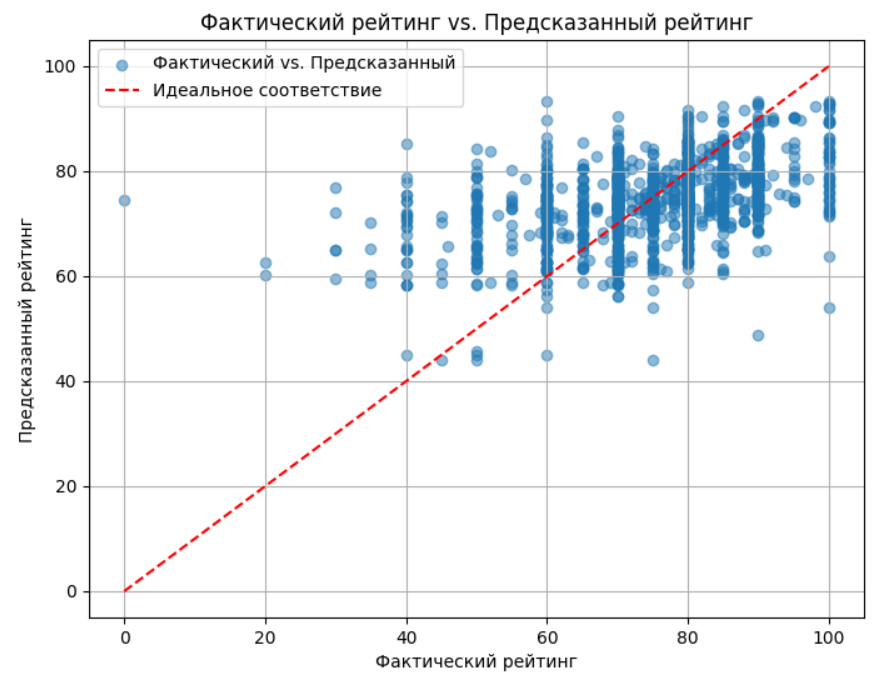


Рисунок 11. Результат выполнения предсказания с помощью нейронной сети MLPRegressor



Рисунок 12. Результат выполнения предсказания с помощью нейронной сети MLPRegressor

Вывод: Сравнив и проанализировав результаты моделей: линейной регрессии c точностью 10.3696 для предсказания и точностью 0.0199 и нейронной сети MLPRegressor с точностью 20.572 на тестовых данных и абсолютным отклонением MAE со значением 27.6 на основе введенных данных при выполнении предсказания, пришел к выводу, что нейронная сеть MLPRegressor более подходит на выполнения задачи предсказания, так как точность модели выше на 10% и нейронная сеть имеет более обширные возможности для улучшения, чтобы повысить точность модели, но по данным результатам ни одна из моделей не являются точными.

## Кластеризация

Проведем кластеризации к данным о играх, для выявления схожих групп игр по признакам. Кластеризации данных игр с использованием моделей linkage с dendrogram и нейронной сети SOM. Краткое описание работы кластеризации с помощью linkage с dendrogram:

Подготовка данных:

* Загрузка данных из файла "games.csv".
* Выбор интересующих переменных для кластеризации, таких как 'genres', 'price', 'platform' и 'developer'.
* Преобразование категориальных переменных в числовые с использованием One-Hot Encoding.

Кластеризация dendrogram:

* Вычисление матрицы расстояний между объектами с использованием метода 'ward'.
* Построение дендрограммы на основе матрицы расстояний.

Создание изображения дендрограммы:

* Создание изображения вручную с использованием библиотек Matplotlib и Seaborn.
* Визуализация данных с помощью тепловой карты, где объекты упорядочены согласно листьям дендрограммы.

Сохранение изображения в формате base64:

* Сохранение изображения в буфер и его преобразование в строку в формате base64.

Определение количества кластеров:

* Использование цветов в дендрограмме для определения количества кластеров.

Веб-интерфейс с Flask:

* Инициализация Flask-приложения с использованием Blueprint.
* Отображение HTML-страницы с дендрограммой и текстовым отчетом на веб-странице.

Отчет:

В текстовом отчете указывается количество обнаруженных кластеров.

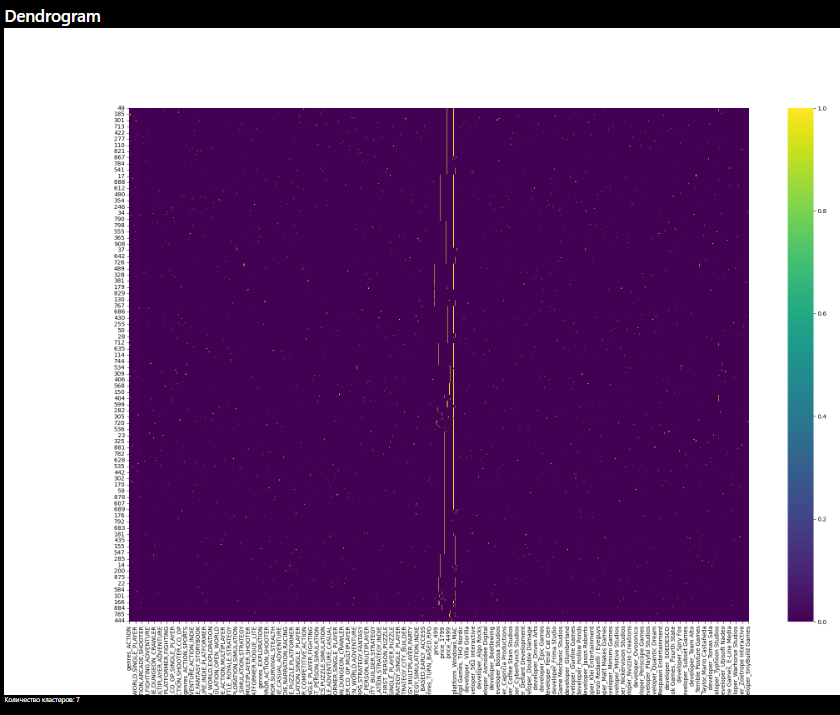


Рисунок 13. Результат выполнения кластеризации с помощью linkage с dendrogram



Рисунок 14. Результат выполнения кластеризации с помощью linkage с dendrogram

Краткое описание работы кластеризации с помощью linkage с dendrogram:

Подготовка данных:

* Загрузка данных из файла "games.csv".
* Выбор интересующих переменных для кластеризации, таких как 'genres', 'price', 'platform' и 'developer'.
* Преобразование категориальных переменных в числовые с использованием One-Hot Encoding.
* Нормализация данных с использованием MinMaxScaler.

Создание и обучение SOM:

* Определение размера карты Кохонена (SOM).
* Создание и обучение SOM с использованием библиотеки MiniSom.

Метки кластеров и визуализация карты кластеров:

* Получение меток кластеров для каждого объекта данных.

Визуализация карты кластеров с использованием цветовой карты 'viridis'.

Оценка качества кластеризации:

* Расчет силуэтной оценки для оценки качества кластеризации.
* Вывод информации о количестве кластеров, размерах кластеров и характеристиках каждого кластера.

Сохранение изображения в формате base64:

* Сохранение изображения карты кластеров в буфер и его преобразование в строку в формате base64.

Веб-интерфейс с Flask:

* Инициализация Flask-приложения с использованием Blueprint.
* Отображение HTML-страницы с картой кластеров и текстовым отчетом на веб-странице.

Отчет:

Текстовый отчет содержит информацию о количестве кластеров, размерах кластеров и характеристиках каждого кластера.

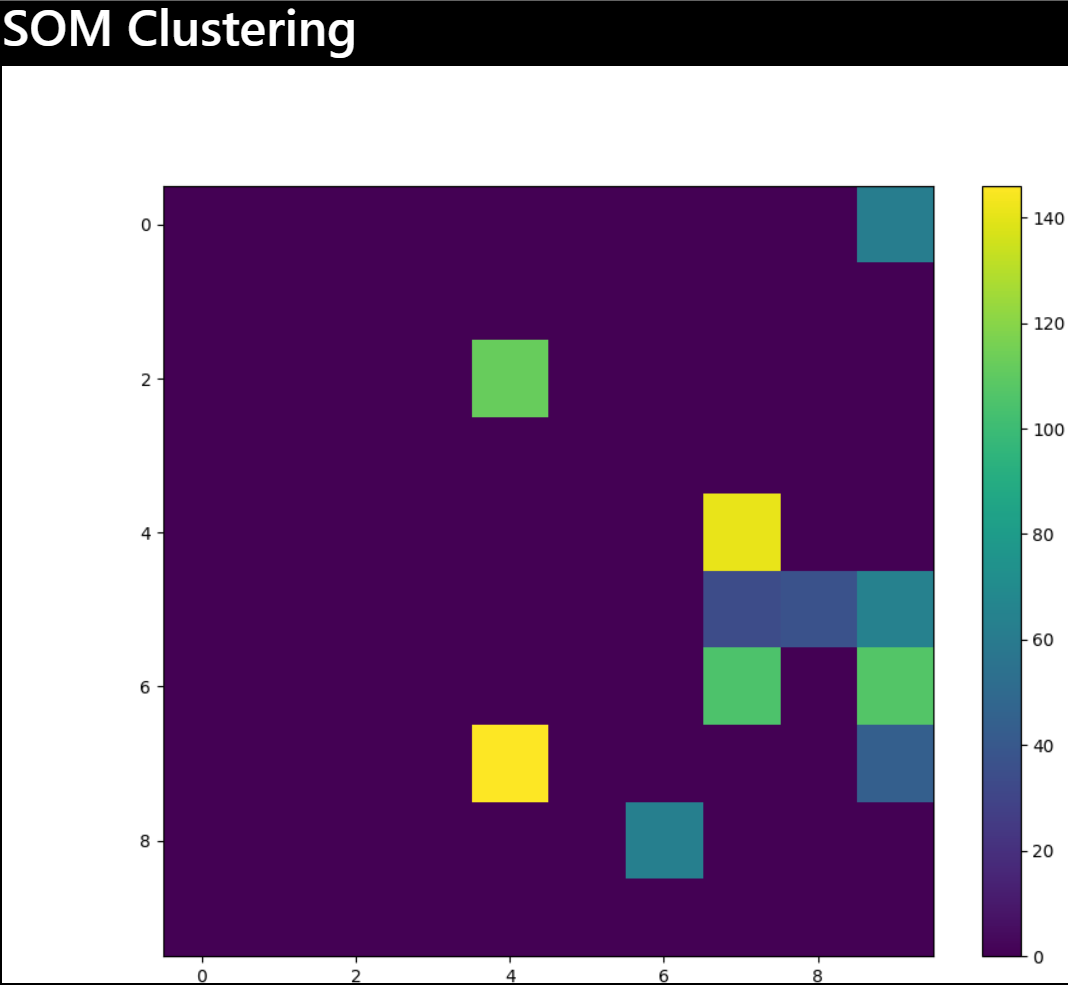


Рисунок 15. Результат выполнения кластеризации с помощью нейронной сети SOM



Рисунок 16. Результат выполнения кластеризации с помощью нейронной сети SOM

## Получение результата

Каждый кластер на графике представлен своим цветом, их количество позволяет сделать вывод о том, какие группы игр имеют схожие характеристики в указанных переменных. Так на основе визуальных результатов по обоим моделям можно сделать вывод, что ни одна из них не подходит под данную задачу, потому что модели плохо работу на больших наборах данных в моем случае почти 20 тысяч записей, так как трудно увидеть и понять, как количество кластеров, так и их размерность по изображению.

# Заключение

В ходе анализа набора данных о видеоиграх на платформе Epic Games Store было проведено исследование с целью выявления факторов, влияющих на успешность игры, прогнозирования успешности игры, такие как логистическая регрессия, деревья решений, многослойный перцептрон, нейронная сеть SOM, линейная регрессия.

По данным моделям получилось, что модели: связей и дендрограммы имеет точность 0.7159, логическая регрессия 0.608, линейная регрессия 10.3696 для предсказания и 0.0199, нейронная сеть MLPRegressor 20.572 и абсолютным отклонением MAE со значением 27.6.

Сравнив модели для задач: классификации, предсказания, кластеризации по сделанным моделям, пришел к выводу, что для классификации дерево решений подходит больше, так как точность модели 0.7159 и она выше, чем точности логической регрессии 0.608. Для предсказания нейронная сеть MLPRegressor более подходит на выполнения задачи предсказания, чем линейная регрессия, так как точность модели выше на 10% и нейронная сеть имеет более обширные возможности для улучшения, чтобы повысить точность модели. Для кластеризации на основе визуальных результатов по моделям нейронной сети SOM и связей с дендрограммой можно сделать вывод, что ни одна из них не подходит под данную задачу, потому что модели плохо работу на больших наборах данных в моем случае почти 20 тысяч записей, так как трудно увидеть и понять, как количество кластеров, так и их размерность по изображению.

Таким образом, на основе проведенного анализа можно сделать вывод о том, что успешность видеоигр на платформе Epic Games Store зависит от определенных характеристик, и группировка игр по их схожим чертам может способствовать более эффективному анализу и пониманию предпочтений пользователей.

# Приложение

Листинг кода

from flask import render\_template, request, Blueprint  
import pandas as pd  
from sklearn.linear\_model import LogisticRegression  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score, confusion\_matrix, roc\_curve, auc  
import matplotlib.pyplot as plt  
import io  
import base64  
  
logical\_regression\_bp = Blueprint('Classification\_Logical\_Regression', \_\_name\_\_)  
  
# Загрузите данные из файлов  
games\_data = pd.read\_csv("games.csv")  
open\_critic\_data = pd.read\_csv("open\_critic.csv")  
  
# Объедините данные  
merged\_data = games\_data.merge(open\_critic\_data, left\_on="id", right\_on="game\_id", how="inner")  
  
# Создайте целевую переменную (успешность игры)  
merged\_data["Success"] = merged\_data["rating"].apply(lambda x: 1 if x > 70 else 0)  
  
# Преобразуйте столбец "release\_date" в формат Unix-времени с корректной обработкой временных зон  
merged\_data["release\_date"] = pd.to\_datetime(merged\_data["release\_date"], utc=True).astype('int64') // 10\*\*9  
  
# Преобразуйте жанры в бинарные признаки (one-hot encoding)  
genres\_encoded = merged\_data['genres'].str.get\_dummies(sep=',')  
merged\_data = pd.concat([merged\_data, genres\_encoded], axis=1)  
merged\_data.drop('genres', axis=1, inplace=True)  
  
# Определите признаки и целевую переменную  
features = ["price", "release\_date"] + list(genres\_encoded.columns)  
X = merged\_data[features]  
y = merged\_data["Success"]  
  
# Разделите данные на обучающий и тестовый наборы  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.01, random\_state=42)  
  
# Создайте и обучите модель логистической регрессии  
model = LogisticRegression()  
model.fit(X\_train, y\_train)  
  
# Оцените модель на тестовом наборе данных  
y\_pred = model.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
  
# Получите матрицу ошибок  
conf\_matrix = confusion\_matrix(y\_test, y\_pred)  
  
# Получите значения ROC-кривой  
fpr, tpr, thresholds = roc\_curve(y\_test, model.predict\_proba(X\_test)[:, 1])  
roc\_auc = auc(fpr, tpr)  
  
# Создание графика ROC-кривой  
plt.figure(figsize=(8, 8))  
plt.plot(fpr, tpr, color='darkorange', lw=2, label='ROC curve (area = {:.2f})'.format(roc\_auc))  
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver Operating Characteristic')  
plt.legend(loc="lower right")  
  
# Сохранение изображения в байтовом массиве  
img\_buf = io.BytesIO()  
plt.savefig(img\_buf, format='png')  
img\_buf.seek(0)  
img\_base64 = base64.b64encode(img\_buf.read()).decode('utf-8')  
plt.close()  
  
# Передача закодированного изображения в шаблон HTML  
img\_tag = f'<img src="data:image/png;base64,{img\_base64}" alt="ROC Curve">'  
  
# Передача данных в шаблон  
@logical\_regression\_bp.route("/", methods=["GET", "POST"])  
def index():  
 img\_tag = None  
 conf\_matrix\_html = None  
 feature\_coefficients = None  
  
 if request.method == "POST":  
 price = float(request.form["price"])  
 release\_date = pd.to\_datetime(request.form["release\_date"], utc=True)  
 genres\_input = request.form["genres"]  
 genres\_encoded\_input = pd.DataFrame(genres\_input.split(','), columns=["genre"])  
 genres\_encoded\_input = genres\_encoded\_input['genre'].str.get\_dummies()  
  
 input\_data = pd.DataFrame(columns=X\_train.columns)  
 input\_data["price"] = [price]  
 release\_date\_unix = release\_date.timestamp()  
 input\_data["release\_date"] = [release\_date\_unix]  
  
 for genre in genres\_encoded\_input.columns:  
 input\_data[genre] = genres\_encoded\_input[genre].values  
  
 # Заполнение пропущенных значений нулями  
 input\_data.fillna(0, inplace=True)  
  
 prediction = model.predict(input\_data)[0]  
 result = "Успешная" if prediction == 1 else "Неуспешная"  
  
 img\_tag = f'<img src="data:image/png;base64,{img\_base64}" alt="ROC Curve">'  
 feature\_coefficients = model.coef\_[0]  
  
 # Преобразование матрицы ошибок в HTML-строку  
 conf\_matrix\_html = f'<p>Матрица ошибок:</p><pre>{conf\_matrix.tolist()}</pre>'  
  
 return render\_template("Classification\_Logical\_Regression.html", accuracy=accuracy, success\_count=sum(y\_test),  
 failure\_count=len(y\_test) - sum(y\_test), conf\_matrix\_html=conf\_matrix\_html,  
 img\_tag=img\_tag, prediction\_result=result, feature\_coefficients=feature\_coefficients)  
  
 return render\_template("Classification\_Logical\_Regression.html", accuracy=accuracy, success\_count=sum(y\_test),  
 failure\_count=len(y\_test) - sum(y\_test), conf\_matrix\_html=conf\_matrix\_html,  
 img\_tag=img\_tag, feature\_coefficients=feature\_coefficients)

from flask import render\_template, Blueprint  
import pandas as pd  
from scipy.cluster.hierarchy import linkage, dendrogram  
import seaborn as sns  
import io  
import base64  
import matplotlib.pyplot as plt  
  
dendrogram\_bp = Blueprint('Clustering\_Linkage\_Dendrogram', \_\_name\_\_)  
  
# Загрузка данных из файла games.csv  
data = pd.read\_csv('games.csv')  
  
# Реализация кластеризации dendrogram  
def generate\_dendrogram\_image():  
 # Выбираем интересующие нас переменные для кластеризации  
 features = data[['genres', 'price', 'platform', 'developer']]  
  
 # Преобразовываем категориальные переменные в числовые (например, с помощью One-Hot Encoding)  
 features = pd.get\_dummies(features, columns=['genres', 'price', 'platform', 'developer'])  
  
 # Вычисляем матрицу расстояний  
 linkage\_matrix = linkage(features, method='ward')  
 dendrogram\_data = dendrogram(linkage\_matrix, no\_plot=True)  
  
 # Создаем изображение вручную с использованием Matplotlib  
 fig, ax = plt.subplots(figsize=(24, 16))  
 g = sns.heatmap(features.iloc[dendrogram\_data['leaves'], :], cmap='viridis', ax=ax)  
 plt.xticks(rotation=90)  
  
 # Сохраняем изображение в буфер  
 img\_buffer = io.BytesIO()  
 fig.savefig(img\_buffer, format='png')  
 img\_buffer.seek(0)  
  
 # Преобразуем изображение в base64 строку  
 img\_base64 = base64.b64encode(img\_buffer.read()).decode()  
  
 # Определяем количество кластеров  
 num\_clusters = len(set(dendrogram\_data['color\_list']))  
  
 # Другие вычисления или данные, которые вы хотите добавить в отчет  
  
 # Формируем текстовый отчет  
 report = f"Количество кластеров: {num\_clusters}\n"  
  
 return img\_base64, report  
  
@dendrogram\_bp.route('/')  
def index():  
 dendrogram\_image, report = generate\_dendrogram\_image()  
 return render\_template('Clustering\_Linkage\_Dendrogram.html', dendrogram\_image=dendrogram\_image, report=report)

from flask import render\_template, Blueprint  
import pandas as pd  
import io  
import base64  
import numpy as np  
from minisom import MiniSom  
import matplotlib.pyplot as plt  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.metrics import silhouette\_score  
  
som\_bp = Blueprint('Clustering\_SOM\_Neural\_Network', \_\_name\_\_)  
  
# Загрузка данных из файла games.csv  
data = pd.read\_csv('games.csv')  
  
# Реализация кластеризации SOM  
def generate\_som\_image():  
 # Выбираем интересующие нас переменные для кластеризации  
 features = data[['genres', 'price', 'platform', 'developer']]  
  
 # Преобразовываем категориальные переменные в числовые (например, с помощью One-Hot Encoding)  
 features = pd.get\_dummies(features, columns=['genres', 'price', 'platform', 'developer'])  
  
 # Нормализуем данные с использованием MinMaxScaler  
 scaler = MinMaxScaler()  
 features\_normalized = scaler.fit\_transform(features)  
  
 # Определяем размер карты Кохонена  
 som\_size = (10, 10) # Подберите подходящий размер  
  
 # Создаем SOM  
 som = MiniSom(som\_size[0], som\_size[1], features.shape[1], sigma=0.3, learning\_rate=0.5)  
  
 # Обучаем SOM  
 som.train\_random(features\_normalized, 1000)  
  
 # Получаем метки кластеров для каждого объекта  
 cluster\_labels = np.array([som.winner(x) for x in features\_normalized])  
  
 # Преобразуем координаты в одномерный массив  
 cluster\_labels\_flat = cluster\_labels[:, 0] \* som\_size[1] + cluster\_labels[:, 1]  
  
 # Создаем карту кластеров  
 cluster\_map = np.zeros((som\_size[0], som\_size[1]))  
  
 for i in range(features.shape[0]):  
 cluster\_map[cluster\_labels[i][0], cluster\_labels[i][1]] += 1  
  
 # Визуализируем карту кластеров  
 plt.figure(figsize=(10, 8))  
 plt.imshow(cluster\_map, cmap='viridis', interpolation='nearest')  
 plt.colorbar()  
  
 # Оцениваем качество кластеризации с использованием метрики силуэта  
 silhouette\_avg = silhouette\_score(features\_normalized, cluster\_labels\_flat)  
  
 # Сохраняем изображение в буфер  
 img\_buffer = io.BytesIO()  
 plt.savefig(img\_buffer, format='png')  
 img\_buffer.seek(0)  
  
 # Преобразуем изображение в base64 строку  
 img\_base64 = base64.b64encode(img\_buffer.read()).decode()  
  
 # Определяем количество уникальных кластеров и их размерность  
 unique\_clusters, cluster\_sizes = np.unique(cluster\_labels\_flat, return\_counts=True)  
 non\_empty\_clusters = unique\_clusters[cluster\_sizes > 0]  
  
 # Размеры кластеров  
 cluster\_sizes = cluster\_sizes[cluster\_sizes > 0]  
  
 # Характеристики кластеров (средние значения)  
 cluster\_features\_mean = [features[cluster\_labels\_flat == i].mean() for i in non\_empty\_clusters]  
  
 # Формируем текстовый отчет  
 report = f"Количество кластеров: {len(non\_empty\_clusters)}\n"  
 report += f"Размеры кластеров: {cluster\_sizes}\n"  
 report += f"\nОценка качества кластеризации (силуэт): {silhouette\_avg:.4f}"  
  
 for i, cluster\_mean in enumerate(cluster\_features\_mean, start=1):  
 report += f"\nХарактеристики кластера {i} (средние значения):\n"  
 report += f"{cluster\_mean}\n"  
  
 return img\_base64, report  
  
@som\_bp.route('/')  
def index():  
 som\_image, report = generate\_som\_image()  
 return render\_template('Clustering\_SOM\_Neural\_Network.html', som\_image=som\_image, report=report)

from flask import render\_template, request, Blueprint  
import pandas as pd  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import accuracy\_score  
import matplotlib.pyplot as plt  
import io  
import base64  
  
decision\_tree\_bp = Blueprint('Decision\_Tree\_Classification', \_\_name\_\_)  
  
# Загрузите данные из файлов  
games\_data = pd.read\_csv("games.csv")  
open\_critic\_data = pd.read\_csv("open\_critic.csv")  
  
# Объедините данные  
merged\_data = games\_data.merge(open\_critic\_data, left\_on="id", right\_on="game\_id", how="inner")  
  
# Создайте целевую переменную (успешность игры)  
merged\_data["Success"] = merged\_data["rating"].apply(lambda x: 1 if x > 70 else 0)  
  
# Преобразуйте столбец "release\_date" в формат Unix-времени с корректной обработкой временных зон  
merged\_data["release\_date"] = pd.to\_datetime(merged\_data["release\_date"], utc=True).astype('int64') // 10\*\*9  
  
# Преобразуйте жанры в бинарные признаки (one-hot encoding)  
genres\_encoded = merged\_data['genres'].str.get\_dummies(sep=',')  
merged\_data = pd.concat([merged\_data, genres\_encoded], axis=1)  
merged\_data.drop('genres', axis=1, inplace=True)  
  
# Определите признаки и целевую переменную  
features = ["price", "release\_date"] + list(genres\_encoded.columns)  
X = merged\_data[features]  
y = merged\_data["Success"]  
  
# Разделите данные на обучающий и тестовый наборы  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.01, random\_state=42)  
  
# Создайте и обучите модель дерева решений  
model = DecisionTreeClassifier()  
model.fit(X\_train, y\_train)  
  
# Оцените модель на тестовом наборе данных  
y\_pred = model.predict(X\_test)  
accuracy = accuracy\_score(y\_test, y\_pred)  
  
# Получите важности признаков  
feature\_importances = model.feature\_importances\_  
  
# Определение индексов признаков, относящихся к категории "жанры", цена и дата релиза  
genres\_indices = [i for i, feature in enumerate(features) if feature in genres\_encoded.columns]  
price\_index = features.index("price")  
release\_date\_index = features.index("release\_date")  
  
# Получение общей важности для категории "жанры", цены и даты релиза  
genres\_importance = feature\_importances[genres\_indices].sum()  
price\_importance = feature\_importances[price\_index]  
release\_date\_importance = feature\_importances[release\_date\_index]  
  
# Создание столбчатой диаграммы для общей важности признаков  
plt.figure(figsize=(10, 3))  
plt.barh(["Genres", "Price", "Release Date"], [genres\_importance, price\_importance, release\_date\_importance])  
plt.xlabel("Importance")  
plt.title("Feature Importance")  
  
# Сохранение изображения в байтовом массиве  
img\_buf = io.BytesIO()  
plt.savefig(img\_buf, format='png')  
img\_buf.seek(0)  
img\_base64 = base64.b64encode(img\_buf.read()).decode('utf-8')  
plt.close()  
  
# Передача закодированного изображения в шаблон HTML  
img\_tag = f'<img src="data:image/png;base64,{img\_base64}" alt="Feature Importance">'  
  
# Передача общей важности для категории "жанры", цены и даты релиза в шаблон HTML  
genres\_importance\_html = f'<p>Общая важность для жанров: {genres\_importance:.4f}</p>'  
price\_importance\_html = f'<p>Важность цены: {price\_importance:.4f}</p>'  
release\_date\_importance\_html = f'<p>Важность даты релиза: {release\_date\_importance:.4f}</p>'  
  
# Передача данных в шаблон  
@decision\_tree\_bp.route("/", methods=["GET", "POST"])  
def index():  
 img\_tag = None  
 genres\_importance\_html = None  
 price\_importance\_html = None  
 release\_date\_importance\_html = None  
 feature\_importances = None  
  
 if request.method == "POST":  
 price = float(request.form["price"])  
 release\_date = pd.to\_datetime(request.form["release\_date"], utc=True)  
 genres\_input = request.form["genres"]  
 genres\_encoded\_input = pd.DataFrame(genres\_input.split(','), columns=["genre"])  
 genres\_encoded\_input = genres\_encoded\_input['genre'].str.get\_dummies()  
  
 input\_data = pd.DataFrame(columns=X\_train.columns)  
 input\_data["price"] = [price]  
 release\_date\_unix = release\_date.timestamp()  
 input\_data["release\_date"] = [release\_date\_unix]  
  
 for genre in genres\_encoded\_input.columns:  
 input\_data[genre] = genres\_encoded\_input[genre].values  
  
 prediction = model.predict(input\_data)[0]  
 result = "Успешная" if prediction == 1 else "Неуспешная"  
  
 img\_tag = f'<img src="data:image/png;base64,{img\_base64}" alt="Feature Importance">'  
 feature\_importances = model.feature\_importances\_  
  
 genres\_importance\_html = f'<p>Общая важность для жанров: {feature\_importances[genres\_indices].sum():.4f}</p>'  
 price\_importance\_html = f'<p>Важность цены: {feature\_importances[price\_index]:.4f}</p>'  
 release\_date\_importance\_html = f'<p>Важность даты релиза: {feature\_importances[release\_date\_index]:.4f}</p>'  
  
 return render\_template("Decision\_Tree\_Classification.html", accuracy=accuracy, success\_count=sum(y\_test),  
 failure\_count=len(y\_test) - sum(y\_test), genres\_importance\_html=genres\_importance\_html,  
 price\_importance\_html=price\_importance\_html,  
 release\_date\_importance\_html=release\_date\_importance\_html, img\_tag=img\_tag,  
 prediction\_result=result, feature\_importances=feature\_importances)  
  
 if feature\_importances is not None:  
 genres\_importance\_html = f'<p>Общая важность для жанров: {feature\_importances[genres\_indices].sum():.4f}</p>'  
 price\_importance\_html = f'<p>Важность цены: {feature\_importances[price\_index]:.4f}</p>'  
 release\_date\_importance\_html = f'<p>Важность даты релиза: {feature\_importances[release\_date\_index]:.4f}</p>'  
  
 return render\_template("Decision\_Tree\_Classification.html", accuracy=accuracy, success\_count=sum(y\_test),  
 failure\_count=len(y\_test) - sum(y\_test), genres\_importance\_html=genres\_importance\_html,  
 price\_importance\_html=price\_importance\_html,  
 release\_date\_importance\_html=release\_date\_importance\_html, img\_tag=img\_tag,  
 feature\_importances=feature\_importances)

from flask import request, render\_template, Blueprint  
import pandas as pd  
from sklearn.preprocessing import LabelEncoder, PolynomialFeatures, StandardScaler  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LinearRegression  
from sklearn.metrics import mean\_absolute\_error, r2\_score  
from datetime import datetime  
import matplotlib.pyplot as plt  
  
linear\_regression\_bp = Blueprint('Prediction\_Linear\_Regression', \_\_name\_\_)  
  
games\_data = pd.read\_csv("games.csv")  
open\_critic\_data = pd.read\_csv("open\_critic.csv")  
  
data = games\_data.merge(open\_critic\_data, left\_on="id", right\_on="game\_id", how="inner")  
data['release\_date'] = (pd.to\_datetime(data['release\_date']) - pd.to\_datetime(data['release\_date']).min()).dt.days  
data = data.dropna(subset=["rating"])  
  
label\_encoder = LabelEncoder()  
data["genres"] = label\_encoder.fit\_transform(data["genres"])  
X = data[["genres", "price", "release\_date"]]  
y = data["rating"]  
  
poly = PolynomialFeatures(degree=3)  
X\_poly = poly.fit\_transform(X[['genres', 'price', 'release\_date']])  
scaler\_poly = StandardScaler()  
X\_poly[:, 3:] = scaler\_poly.fit\_transform(X\_poly[:, 3:])  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.2, random\_state=42)  
  
regressor = LinearRegression()  
regressor.fit(X\_train, y\_train)  
  
min\_release\_date = min(pd.to\_datetime(games\_data['release\_date'])).replace(tzinfo=None)  
  
@linear\_regression\_bp.route('/', methods=['GET'])  
def index():  
 return render\_template('Prediction\_Linear\_Regression.html')  
  
@linear\_regression\_bp.route('/predict', methods=['POST'])  
def predict():  
 data = request.form  
 genre = label\_encoder.transform([data["genres"]])[0]  
 price = float(data["price"])  
  
 release\_date = datetime.strptime(data["release\_date"], "%Y-%m-%dT%H:%M:%S.%fZ")  
 release\_date = (release\_date - min\_release\_date).days  
  
 input\_data = poly.transform([[genre, price, release\_date]])  
 input\_data[:, 3:] = scaler\_poly.transform(input\_data[:, 3:])  
  
 rating = regressor.predict(input\_data)[0]  
  
 y\_pred = regressor.predict(X\_test)  
 prediction\_accuracy = mean\_absolute\_error(y\_test, y\_pred)  
 model\_accuracy = r2\_score(y\_test, y\_pred)  
  
 plt.figure(figsize=(12, 6))  
 plt.scatter(y\_test, y\_pred, color='blue', label='Фактический vs. Предсказанный')  
 plt.plot([min(y\_test), max(y\_test)], [min(y\_test), max(y\_test)], linestyle='--', color='red', label='Идеальное соответствие')  
 plt.title('Фактический vs. Предсказанный рейтинг')  
 plt.xlabel('Фактический рейтинг')  
 plt.ylabel('Предсказанный рейтинг')  
 plt.legend()  
 plt.savefig('static/scatter\_plot.png')  
 plt.close()  
  
 residuals = y\_test - y\_pred  
 plt.figure(figsize=(12, 6))  
 plt.hist(residuals, bins=30, edgecolor='black')  
 plt.title('Распределение остатков')  
 plt.xlabel('Остатки')  
 plt.ylabel('Частота')  
 plt.savefig('static/residuals\_histogram.png')  
 plt.close()  
  
 return render\_template('Prediction\_Linear\_Regression.html', rating=rating, prediction\_accuracy=prediction\_accuracy, model\_accuracy=model\_accuracy)

import pandas as pd  
from flask import render\_template, request, Blueprint  
from sklearn.neural\_network import MLPRegressor  
from sklearn.model\_selection import train\_test\_split  
from sklearn.preprocessing import MinMaxScaler  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error  
import matplotlib.pyplot as plt  
from io import BytesIO  
import base64  
  
mlp\_regressor\_bp = Blueprint('Prediction\_Neural\_Network\_MLPRegressor', \_\_name\_\_)  
  
# Загрузка данных из файлов  
games\_df = pd.read\_csv('games.csv')  
open\_critic\_df = pd.read\_csv('open\_critic.csv')  
  
# Объединение данных по столбцам id и game\_id  
merged\_df = pd.merge(games\_df, open\_critic\_df, left\_on='id', right\_on='game\_id')  
  
# Проверка и удаление строк с NaN значениями в X и y  
merged\_df = merged\_df.dropna(subset=['genres', 'price', 'release\_date', 'rating'])  
  
# Выбор нужных столбцов для обучения  
X = merged\_df[['genres', 'price', 'release\_date']]  
y = merged\_df['rating']  
  
# Преобразование категориальных признаков в числовые  
X = pd.get\_dummies(X, columns=['genres'])  
  
# Преобразование даты релиза в числовой формат (в этом примере используется год релиза)  
X['release\_year'] = pd.to\_datetime(X['release\_date']).dt.year  
X = X.drop(['release\_date'], axis=1)  
  
# Разделение данных на обучающий и тестовый набор  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.1, random\_state=42)  
  
# Нормализация данных  
scaler = MinMaxScaler()  
X\_train\_scaled = scaler.fit\_transform(X\_train)  
X\_test\_scaled = scaler.transform(X\_test)  
  
# Инициализация и обучение MLPRegressor  
mlp\_regressor = MLPRegressor(  
 hidden\_layer\_sizes=(200, 100),  
 activation='relu',  
 max\_iter=1000,  
 solver='adam',  
 alpha=0.01,  
 random\_state=42  
)  
  
mlp\_regressor.fit(X\_train\_scaled, y\_train)  
  
# Получение точности на тестовых данных  
accuracy = mlp\_regressor.score(X\_test\_scaled, y\_test) \* 100  
  
  
# Маршрут для отображения HTML-страницы  
@mlp\_regressor\_bp.route('/')  
def index():  
 # Получение стандартного отклонения на тестовых данных  
 std\_deviation = y\_test.std()  
  
 # Создание графика  
 plt.figure(figsize=(8, 6))  
 plt.scatter(y\_test, mlp\_regressor.predict(X\_test\_scaled), alpha=0.5, label='Фактический vs. Предсказанный')  
  
 # Добавление идеальной диагональной линии  
 plt.plot([y\_test.min(), y\_test.max()], [y\_test.min(), y\_test.max()], color='red', linestyle='--',  
 label='Идеальное соответствие')  
  
 plt.title('Фактический рейтинг vs. Предсказанный рейтинг')  
 plt.xlabel('Фактический рейтинг')  
 plt.ylabel('Предсказанный рейтинг')  
 plt.legend()  
 plt.grid(True)  
  
 # Сохранение графика в байтовый объект  
 image\_stream = BytesIO()  
 plt.savefig(image\_stream, format='png')  
 plt.close()  
  
 # Кодирование изображения в формат base64 для вставки в HTML  
 image\_stream.seek(0)  
 encoded\_image = base64.b64encode(image\_stream.read()).decode()  
  
 return render\_template('Prediction\_Neural\_Network\_MLPRegressor.html', accuracy=accuracy, std\_deviation=std\_deviation, plot\_image=encoded\_image)  
  
# Маршрут для предсказания рейтинга на основе входных данных  
@mlp\_regressor\_bp.route('/predict', methods=['POST'])  
def predict():  
 data = request.form.to\_dict()  
  
 # Преобразование входных данных в формат, подходящий для модели  
 input\_data = pd.DataFrame(data, index=[0])  
 input\_data['price'] = int(input\_data['price'])  
 input\_data['release\_year'] = pd.to\_datetime(input\_data['releaseDate']).dt.year  
 input\_data = pd.get\_dummies(input\_data, columns=['genres'])  
  
 # Удаление releaseDate, так как она больше не нужна после преобразования  
 input\_data = input\_data.drop(['releaseDate'], axis=1)  
  
 # Добавление отсутствующих дамми-переменных  
 missing\_cols = set(X.columns) - set(input\_data.columns)  
 for col in missing\_cols:  
 input\_data[col] = 0  
  
 # Упорядочивание столбцов в нужном порядке  
 input\_data = input\_data[X.columns]  
  
 input\_data\_scaled = scaler.transform(input\_data)  
  
 # Предсказание рейтинга  
 prediction = mlp\_regressor.predict(input\_data\_scaled)  
  
 # Оценка точности на новых данных  
 mae\_on\_prediction = mean\_absolute\_error([y\_test.iloc[0]], [prediction[0]])  
 mse\_on\_prediction = mean\_squared\_error([y\_test.iloc[0]], [prediction[0]])  
  
 return render\_template('Prediction\_Neural\_Network\_MLPRegressor.html', prediction\_result=f'Predicted Rating: {prediction[0]:.2f}',  
 mae\_on\_prediction=f'MAE on Prediction: {mae\_on\_prediction:.2f}',  
 mse\_on\_prediction=f'MSE on Prediction: {mse\_on\_prediction:.2f}')

<!DOCTYPE html>  
<html lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <style>  
 body {  
 background-color: #000000;  
 color: #ffffff;  
 }  
  
 #content {  
 margin: 20px;  
 }  
 </style>  
 <title>Классификация игр</title>  
</head>  
<body>  
 <h1>Классификация игр</h1>  
  
 <!-- Форма для ввода данных -->  
 <form method="POST" action="/Classification\_Logical\_Regression/">  
 <label for="price">Цена игры:</label>  
 <input type="text" name="price" id="price" required>  
 <br>  
 <label for="genres">Жанры игры:</label>  
 <input type="text" name="genres" id="genres" required>  
 <br>  
 <label for="release\_date">Дата релиза игры:</label>  
 <input type="text" name="release\_date" id="release\_date" required>  
 <br>  
 <button type="submit">Выполнить</button>  
 </form>  
  
 <!-- Результат классификации -->  
 {% if prediction\_result %}  
 <h2>Результат классификации:</h2>  
 <p>Игра классифицирована как: {{ prediction\_result }}</p>  
 {% endif %}  
  
 <!-- Другие результаты (точность, количество успешных и неуспешных игр) -->  
 <h2>Другие результаты:</h2>  
 <p>Точность модели: {{ "%.4f"|format(accuracy) }}</p>  
 <p>Количество успешных игр: {{ success\_count }}</p>  
 <p>Количество неуспешных игр: {{ failure\_count }}</p>  
  
 <!-- График ROC-кривой -->  
 {% if img\_tag %}  
 <h2>ROC-кривая:</h2>  
 {{ img\_tag|safe }}  
 {% endif %}  
  
 <!-- Матрица ошибок -->  
 {% if conf\_matrix\_html %}  
 <h2>Матрица ошибок:</h2>  
 {{ conf\_matrix\_html|safe }}  
 {% endif %}  
</body>  
</html>

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>Dendrogram</title>  
</head>  
<body>  
 <h1>Dendrogram</h1>  
 <img src="data:image/png;base64,{{ dendrogram\_image }}" alt="Dendrogram">  
 <p>{{ report }}</p>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <title>SOM Clustering</title>  
</head>  
<body>  
 <h1>SOM Clustering</h1>  
 <img src="data:image/png;base64,{{ som\_image }}" alt="SOM Clustering">  
 <p>{{ report }}</p>  
</body>  
</html>

<!DOCTYPE html>  
<html lang="ru">  
<head>  
 <meta charset="UTF-8">  
 <style>  
 body {  
 background-color: #000000;  
 color: #ffffff;  
 }  
  
 #content {  
 margin: 20px;  
 }  
 </style>  
 <title>Классификация игр</title>  
</head>  
<body>  
 <h1>Классификация игр</h1>  
  
 <!-- Форма для ввода данных -->  
 <form method="POST" action="/Decision\_Tree\_Classification/">  
 <label for="price">Цена игры:</label>  
 <input type="text" name="price" id="price" required>  
 <br>  
 <label for="genres">Жанры игры:</label>  
 <input type="text" name="genres" id="genres" required>  
 <br>  
 <label for="release\_date">Дата релиза игры:</label>  
 <input type="text" name="release\_date" id="release\_date" required>  
 <br>  
 <button type="submit">Выполнить</button>  
 </form>  
  
 <!-- Результат классификации -->  
 {% if prediction\_result %}  
 <h2>Результат классификации:</h2>  
 <p>Игра классифицирована как: {{ prediction\_result }}</p>  
 {% endif %}  
  
 <!-- Другие результаты (точность, количество успешных и неуспешных игр, важности признаков) -->  
 <h2>Другие результаты:</h2>  
 <p>Точность модели: {{ "%.4f"|format(accuracy) }}</p>  
 <p>Количество успешных игр: {{ success\_count }}</p>  
 <p>Количество неуспешных игр: {{ failure\_count }}</p>  
  
 {% if genres\_importance\_html %}  
 <h2>Важность признаков:</h2>  
 {{ genres\_importance\_html | safe }}  
 <p>Важность цены: {{ "%.4f"|format(feature\_importances[0]) }}</p>  
 <p>Важность даты релиза: {{ "%.4f"|format(feature\_importances[1]) }}</p>  
 {% endif %}  
  
 <!-- График важности признаков -->  
 {% if img\_tag %}  
 <h2>График важности признаков:</h2>  
 {{ img\_tag | safe }}  
 {% endif %}  
</body>  
</html>

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/css/bootstrap.min.css" rel="stylesheet">  
 <title>Курсовая работа по ИИС</title>  
 <style>  
 body {  
 background-color: #000000;  
 color: #ffffff;  
 }  
  
 .navbar {  
 background-color: #000000;  
 box-shadow: 0 4px 6px -1px rgba(0,0,0,0.1), 0 2px 4px -1px rgba(0,0,0,0.06);  
 }  
  
 .navbar-brand, .navbar-nav .nav-link {  
 color: #ffffff;  
 }  
  
 .navbar-toggler-icon {  
 background-color: #ffffff;  
 }  
  
 #content {  
 margin: 20px;  
 }  
  
 .dragon-logo {  
 width: 100px;  
 height: auto;  
 }  
  
 .dropdown-menu {  
 background-color: #000000;  
 }  
  
 .dropdown-item {  
 color: #ffffff;  
 }  
  
 .dropdown-item:hover {  
 background-color: #007bff;  
 }  
  
 .center-image {  
 text-align: center;  
 margin-top: 20px;  
 }  
  
 .navbar-brand:hover {  
 color: #ffffff; /\* Цвет при наведении на ссылку \*/  
 }  
 </style>  
</head>  
<body>  
  
<nav class="navbar navbar-expand-lg navbar-dark">  
 <div class="container-fluid">  
 <a class="navbar-brand" href="https://vk.com/sav.alex9" target="\_blank">  
 <img src="{{ url\_for('static', filename='dragon\_logo.png') }}" alt="Dragon Logo" class="dragon-logo">  
 </a>  
 <a class="navbar-brand" href="http://localhost:5000/">  
 Курсовая работа по ИИС  
 </a>  
 <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav" aria-controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">  
 <span class="navbar-toggler-icon"></span>  
 </button>  
 <div class="collapse navbar-collapse" id="navbarNav">  
 <ul class="navbar-nav">  
 <li class="nav-item dropdown">  
 <a class="nav-link dropdown-toggle" href="#" id="classificationDropdown" role="button" data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
 Классификация  
 </a>  
 <div class="dropdown-menu" aria-labelledby="classificationDropdown">  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Classification\_Logical\_Regression')">Логистическая регрессия</a>  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Decision\_Tree\_Classification')">Дерево решений</a>  
 </div>  
 </li>  
 <li class="nav-item dropdown">  
 <a class="nav-link dropdown-toggle" href="#" id="clusteringDropdown" role="button" data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
 Кластеризация  
 </a>  
 <div class="dropdown-menu" aria-labelledby="clusteringDropdown">  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Clustering\_Linkage\_Dendrogram')">Связь и дендрограмма</a>  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Clustering\_SOM\_Neural\_Network')">SOM</a>  
 </div>  
 </li>  
 <li class="nav-item dropdown">  
 <a class="nav-link dropdown-toggle" href="#" id="predictionDropdown" role="button" data-bs-toggle="dropdown" aria-haspopup="true" aria-expanded="false">  
 Предсказание  
 </a>  
 <div class="dropdown-menu" aria-labelledby="predictionDropdown">  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Prediction\_Linear\_Regression')">Линейная регрессия</a>  
 <a class="dropdown-item" href="#" onclick="loadSubContent('Prediction\_Neural\_Network\_MLPRegressor')">MLPRegressor</a>  
 </div>  
 </li>  
 </ul>  
 </div>  
 </div>  
</nav>  
  
<div id="content"></div>  
  
<div class="center-image">  
 <a href="https://vk.com/club215754315" target="\_blank">  
 <img src="{{ url\_for('static', filename='center\_image.png') }}" alt="Center Image">  
 </a>  
</div>  
  
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.0.2/dist/js/bootstrap.bundle.min.js"></script>  
<script>  
 function loadSubContent(subSection) {  
 fetch('/' + subSection) // Делаем запрос на сервер  
 .then(response => response.text()) // Получаем текст ответа  
 .then(content => {  
 document.getElementById('content').innerHTML = content; // Заменяем содержимое элемента  
 })  
 .catch(error => console.error('Error:', error));  
 }  
</script>  
  
</body>  
</html>

<!DOCTYPE html>  
<html>  
<head>  
 <style>  
 body {  
 background-color: #000000;  
 color: #ffffff;  
 }  
  
 #content {  
 margin: 20px;  
 }  
 </style>  
 <title>Предсказание рейтинга игры</title>  
</head>  
<body>  
 <h1>Предсказание рейтинга игры</h1>  
 <form method="post" action="/Prediction\_Linear\_Regression/predict">  
 <label for="genres">Жанр:</label>  
 <input type="text" name="genres" id="genres" required><br>  
  
 <label for="price">Цена:</label>  
 <input type="number" name="price" id="price" required><br>  
  
 <label for="release\_date">Дата релиза:</label>  
 <input type="text" name="release\_date" id="release\_date" required><br>  
  
 <input type="submit" value="Предсказать рейтинг">  
 </form>  
  
 {% if rating is defined %}  
 <h2>Предсказанный рейтинг игры: {{ rating|round(2) }}</h2>  
 {% endif %}  
  
 {% if prediction\_accuracy is defined %}  
 <h2>Точность предсказания: {{ prediction\_accuracy|round(4) }}</h2>  
 {% endif %}  
  
 {% if model\_accuracy is defined %}  
 <h2>Точность модели по тестовым данным: {{ model\_accuracy|round(4) }}</h2>  
 {% endif %}  
  
 {% if rating is defined %}  
 <h2>График фактических и предсказанных значений</h2>  
 <img src="{{ url\_for('static', filename='scatter\_plot.png') }}" alt="График фактических и предсказанных значений">  
  
 <h2>График распределения остатков</h2>  
 <img src="{{ url\_for('static', filename='residuals\_histogram.png') }}" alt="График распределения остатков">  
 {% endif %}  
  
</body>  
</html>

<!DOCTYPE html>  
<html lang="en">  
<head>  
 <meta charset="UTF-8">  
 <meta name="viewport" content="width=device-width, initial-scale=1.0">  
 <style>  
 body {  
 background-color: #000000;  
 color: #ffffff;  
 }  
  
 #content {  
 margin: 20px;  
 }  
 </style>  
 <title>Предсказания рейтинга игр</title>  
</head>  
<body>  
 <h1>Предсказания рейтинга игр</h1>  
  
 <form action="/Prediction\_Neural\_Network\_MLPRegressor/predict" method="post">  
 <label for="genres">Жанры:</label>  
 <input type="text" id="genres" name="genres" required><br>  
  
 <label for="price">Цена:</label>  
 <input type="number" id="price" name="price" required><br>  
  
 <label for="releaseDate">Дата релиза:</label>  
 <input type="text" id="releaseDate" name="releaseDate" placeholder="YYYY-MM-DD" required><br>  
  
 <button type="submit">Предсказать рейтинг</button>  
 </form>  
  
 {% if prediction\_result %}  
 <h2>Результат предсказания:</h2>  
 <p>{{ prediction\_result }}</p>  
 {% endif %}  
  
 {% if mae\_on\_prediction %}  
 <h2>MAE на предсказании:</h2>  
 <p>{{ mae\_on\_prediction }}</p>  
 {% endif %}  
  
 {% if mse\_on\_prediction %}  
 <h2>MSE на предсказании:</h2>  
 <p>{{ mse\_on\_prediction }}</p>  
 {% endif %}  
  
 {% if accuracy %}  
 <h2>Точность модели:</h2>  
 <p>{{ accuracy }}% (on test data)</p>  
 {% endif %}  
  
 {% if std\_deviation %}  
 <h2>Стандартное отклонение рейтингов:</h2>  
 <p>{{ std\_deviation }}% (on test data)</p>  
 {% endif %}  
  
 {% if plot\_image %}  
 <h2>График предсказаний:</h2>  
 <img src="data:image/png;base64, {{ plot\_image }}" alt="График предсказаний">  
 {% endif %}  
  
</body>  
</html>

from flask import Flask, render\_template  
from Machine\_Learning\_And\_Neural\_Network\_Models.Classification\_Logical\_Regression.Classification\_Logical\_Regression import logical\_regression\_bp  
from Machine\_Learning\_And\_Neural\_Network\_Models.Decision\_Tree\_Classification.Decision\_Tree\_Classification import decision\_tree\_bp  
from Machine\_Learning\_And\_Neural\_Network\_Models.Clustering\_Linkage\_Dendrogram.Clustering\_Linkage\_Dendrogram import dendrogram\_bp  
from Machine\_Learning\_And\_Neural\_Network\_Models.Clustering\_SOM\_Neural\_Network.Clustering\_SOM\_Neural\_Network import som\_bp  
from Machine\_Learning\_And\_Neural\_Network\_Models.Prediction\_Linear\_Regression.Prediction\_Linear\_Regression import linear\_regression\_bp  
from Machine\_Learning\_And\_Neural\_Network\_Models.Prediction\_Neural\_Network\_MLPRegressor.Prediction\_Neural\_Network\_MLPRegressor import mlp\_regressor\_bp  
  
app = Flask(\_\_name\_\_)  
  
# Регистрация Blueprint'ов  
app.register\_blueprint(logical\_regression\_bp, url\_prefix='/Classification\_Logical\_Regression')  
app.register\_blueprint(decision\_tree\_bp, url\_prefix='/Decision\_Tree\_Classification')  
app.register\_blueprint(dendrogram\_bp, url\_prefix='/Clustering\_Linkage\_Dendrogram')  
app.register\_blueprint(som\_bp, url\_prefix='/Clustering\_SOM\_Neural\_Network')  
app.register\_blueprint(linear\_regression\_bp, url\_prefix='/Prediction\_Linear\_Regression')  
app.register\_blueprint(mlp\_regressor\_bp, url\_prefix='/Prediction\_Neural\_Network\_MLPRegressor')  
  
# Главная страница  
@app.route("/")  
def home():  
 return render\_template("home.html")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 app.run(host="localhost", port=5000)

# Список литературы

1. Epic Games Store Dataset [Электронный ресурс]: Kaggle. — Электрон. дан. — Режим доступа: https://www.kaggle.com/datasets/mexwell/epic-games-store-dataset?select=games.csv — Загл. с экрана.
2. Scikit-learn documentation [Электронный ресурс]: Scikit-learn. — Электрон. дан. — Режим доступа: https://scikit-learn.org/stable/index.html. — Загл. с экрана.
3. Pandas documentation [Электронный ресурс]: pandas. — Электрон. дан. — Режим доступа: https://pandas.pydata.org/pandas-docs/stable/index.html. — Загл. с экрана.
4. NumPy Documentation [Электронный ресурс]: NumPy. — Электрон. дан. — Режим доступа: https://numpy.org/doc/. — Загл. с экрана.
5. Теория и практика машинного обучения : учебное пособие / В. В. Воронина, А. В. Михеев, Н. Г. Ярушкина, К. В. Святов. – Ульяновск : УлГТУ, 2017. – 290 с.