

НАЦИОНАЛЬНЫЙ  
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ  
ВЫСШАЯ ШКОЛА ЭКОНОМИКИ

Факультет компьютерных наук  
Магистерская программа: “Науки о данных”

**Отчет по проекту**

по теме ”Построение системы музыкальных рекомендаций при помощи методов машинного обучения”

**Выполнена студентом:**

группы МНОД 231 \_\_\_\_\_ Шкляром Михаилом Игоревичем  
Фамилия, Имя, Отчество

**Проверено научным руководителем проекта:**

Николенко Сергеем Игоревичем

Фамилия, Имя, Отчество

Доцент

Ученая степень

Департамент анализа данных и  
искусственного интеллекта

Место работы (Департамент ВШЭ)

**Москва 2024**

# Содержание

<b>Annotation.....</b>	<b>4</b>
<b>1. Введение.....</b>	<b>5</b>
1.1 Описание предметной области.....	5
1.2 Постановка задачи.....	5
<b>2. Обзор аналогов и литературы.....</b>	<b>6</b>
2.1 Анализ конкурентов.....	6
2.2 Анализ публикаций.....	7
2.2.1 Публикация: "Deep Content-based Music Recommendation".....	7
2.2.1 Публикация: "Collaborative Filtering for Music Recommendation".....	7
2.2.3 Выводы по публикациям.....	7
<b>3. Сбор данных.....</b>	<b>8</b>
3.1 Парсинг данных со Spotify.....	8
3.1.1 Пример с жанром hip-hop:.....	8
3.2 Объединение треков в единый датасет.....	9
<b>4. Препроцессинг.....</b>	<b>11</b>
4.1 Применение StandardScaler.....	11
4.2 Попытка логарифмирования.....	12
4.3 Удаление выбросов.....	12
<b>5. Анализ данных.....</b>	<b>12</b>
5.1 Анализ результатов.....	14
<b>6. Машинное обучение.....</b>	<b>15</b>
6.1 CatBoost.....	15
6.2 XGBoost.....	17
6.2.1 Настройка и оценка модели.....	17
6.2.2 Важность признаков.....	17
6.3 LightBLM.....	18
6.4 SVM.....	19
<b>7. Построение нейронной сети.....</b>	<b>19</b>
7.1 Используемые методы.....	19
7.2 Подготовка данных и инициализация модели.....	20
7.3 Компиляция и обучение модели.....	21
7.4 Оценка и валидация модели.....	21
<b>8. Выводы по обученным моделям.....</b>	<b>23</b>
<b>9. Telegram-bot.....</b>	<b>23</b>
9.1 Основные функции бота:.....	24
9.2 Получение ссылки на трек.....	24
9.3 Запрос данных о треке.....	24
9.4 Анализ и предсказание жанра.....	25
9.5 Поиск видео на YouTube.....	25
9.6 Рекомендация похожего трека.....	25
9.7 Взаимодействие и завершение.....	25
<b>10. Web-приложение.....</b>	<b>26</b>
10.1 Введение.....	26

10.2 Архитектура приложения.....	27
10.3 Разработка.....	28
10.3.1 Установка и настройка окружения.....	28
10.3.1 Загрузка модели и данных.....	29
10.3.2 Настройка API для работы с музыкальными сервисами.....	29
10.4 Пользовательский интерфейс.....	32
10.5 Развёртывание.....	33
<b>10.6 Тестирование.....</b>	<b>34</b>
List of Sources.....	36

## **Аннотация**

В данной курсовой работе рассматривается разработка системы музыкальных рекомендаций с использованием методов машинного и глубокого обучения. Основной целью является создание модели, которая способна предсказывать музыкальный жанр на основе аудио характеристик трека. Работа включает сбор данных через Spotify API, обработку и анализ полученных данных, обучение нескольких моделей машинного обучения, включая CatBoost, и нейросети. Наилучшей для интеграции в веб-приложение и телеграм-бот для рекомендаций. Модель CatBoost показала наилучшие результаты, поэтому была использована для интеграции в веб-приложение и телеграм-бот для рекомендаций. Реализованное приложение позволяет пользователям получать музыкальные рекомендации по введенному названию трека или ссылке на Spotify, а также наслаждаться предложенными треками через интегрированные плееры Spotify и YouTube.

## **Annotation**

This thesis examines the development of a music recommendation system using both machine learning and deep learning methods. The primary objective is to create a model capable of predicting the musical genre based on the audio characteristics of a track. The work includes collecting data via the Spotify API, processing and analyzing the data, and training several machine learning models, including CatBoost and neural networks. The CatBoost model demonstrated the best results and was chosen for integration into a web application and a Telegram bot for recommendations. The implemented application allows users to receive music recommendations by entering a track name or a Spotify link and to enjoy the suggested tracks through integrated Spotify and YouTube players.

## **Ключевые слова:**

Система музыкальных рекомендаций, Машинное обучение, CatBoost, Нейронные сети, Spotify API, Классификация музыкальных жанров, Веб-приложение, Телеграм-бот.

# **1. Введение**

## **1.1 Описание предметной области**

Музыкальные рекомендации играют важную роль в мире цифрового контента, где каждый пользователь стремится найти что-то новое и интересное для прослушивания. В условиях растущего количества музыкальных треков и жанров, задача автоматической классификации и предложения музыки становится особенно актуальной. Музыкальные сервисы, такие как Spotify и Apple Music, используют сложные алгоритмы, чтобы предлагать пользователям песни и артистов на основе их предыдущих предпочтений.

В последние годы, с развитием технологий искусственного интеллекта и машинного обучения, возможности в области музыкальных рекомендаций значительно расширились. Это позволяет не только улучшить пользовательский опыт, но и способствует более глубокому и персонализированному взаимодействию с музыкальным контентом.

## **1.2 Постановка задачи**

В рамках данной курсовой работы рассматривается задача разработки и реализации различных моделей машинного обучения для предсказания музыкальных жанров на основе аудио характеристик треков. Основная цель заключается в том, чтобы с высокой точностью идентифицировать жанровую принадлежность музыкальных композиций с минимальной задержкой, что критически важно для предоставления актуальных и интересных музыкальных предложений пользователям.

В дополнение к классическим моделям машинного обучения особое внимание уделялось реализации нейронных сетей. Одним из основных задач нейросетей было сжатие данных в скрытое пространство с последующей реконструкцией входной информации, что особенно полезно для работы с многомерными наборами данных. Преимущество использования такой модели заключается в том, что она включает несупервизируемое обучение, что особенно важно в условиях отсутствия меток.

Кроме того, в проект также включена разработка API, как продукта, позволяющего интегрировать модели с лучшими показателями в существующие информационные системы музыкальных сервисов.

## **2. Обзор аналогов и литературы**

### **2.1 Анализ конкурентов**

На фоне таких крупных игроков, как Яндекс Музыка и Spotify, наша система музыкальных рекомендаций выделяется уникальным набором функций и высоким качеством предсказаний. Одним из ключевых преимуществ нашего подхода является способность интеграции с видеоплатформами, что позволяет пользователям не только слушать музыку, но и просматривать музыкальные клипы и живые выступления. Это значительно расширяет пользовательский опыт по сравнению с Яндекс Музыкой, где такая возможность отсутствует.

В отличие от Spotify, который хоть и предоставляет широкий спектр музыкального контента, наша система позволяет начать рекомендации с любой конкретной песни, выбранной пользователем. Это добавляет гибкости в процесс выбора музыкальных треков и делает предложения более целенаправленными и личностно значимыми. К тому же, наша модель показывает высокую точность в классификации музыкальных жанров, достигая 69.03%, что является впечатляющим результатом для классификации множества жанров.

Еще одним значимым аспектом нашего проекта является создание телеграм-бота, который предоставляет возможность управления музыкальными рекомендациями прямо из мессенджера. Это предоставляет дополнительное удобство для пользователей, которое не доступно в Яндекс Музыке и Spotify. Наш подход позволяет обойти географические и лицензионные ограничения, с которыми сталкивается Яндекс Музыка, предлагая доступ ко всему мировому репертуару без ограничений.

Благодаря использованию открытых источников данных и продвинутых технологий машинного обучения, наша система способна предложить не только широкий спектр музыкальных жанров, но и высокую точность в их предсказании. Это делает нашу систему рекомендаций не только конкурентоспособной, но и предпочтительной для пользователей, стремящихся к более глубокому и разнообразному музыкальному опыту.

## **2.2 Анализ публикаций**

### **2.2.1 Публикация: "Deep Content-based Music Recommendation"**

В статье "Deep Content-based Music Recommendation" исследуются методы глубокого обучения для создания рекомендательных систем в музыкальной индустрии. Авторы фокусируются на использовании сверточных нейронных сетей (CNN) для анализа спектрограмм музыкальных треков, что позволяет модели изучать музыкальные предпочтения на основе содержания треков, а не метаданных или пользовательских взаимодействий. Это подход позволяет достигать высокой точности в предсказании предпочтений пользователя, особенно в случаях, когда доступные метаданные ограничены или отсутствуют.

Однако, как отмечают авторы, использование только содержательного анализа может не учитывать социальные и контекстные аспекты музыкального восприятия, такие как популярность треков или сезонные предпочтения слушателей. Это подчеркивает важность комбинирования различных подходов в создании музыкальных рекомендаций.

### **2.2.1 Публикация: "Collaborative Filtering for Music Recommendation"**

В другой значимой работе, "Collaborative Filtering for Music Recommendation", авторы анализируют применение методов коллаборативной фильтрации для музыкальных рекомендаций. Исследование подчеркивает, как совместное использование данных пользовательских оценок может помочь в выявлении скрытых паттернов в музыкальных предпочтениях, что способствует формированию более точных рекомендаций.

Статья также освещает проблемы, связанные с масштабированием коллаборативной фильтрации, включая проблемы холодного старта и разреженности данных. Авторы предлагают использование гибридных систем, которые объединяют коллаборативную фильтрацию с контент-ориентированными подходами для улучшения качества рекомендаций.

### **2.2.3 Выводы по публикациям**

Обе статьи подчеркивают значимость и эффективность машинного обучения и глубокого обучения в задачах музыкальных рекомендаций. В то время как первая статья демонстрирует силу содержательного анализа через глубокое обучение для улучшения точности музыкальных рекомендаций, вторая работа обращает внимание на значение коллаборативной фильтрации в понимании пользовательских взаимодействий.

Для нашей системы музыкальных рекомендаций, совмещение этих подходов может предложить лучшие стратегии для достижения высокой точности и удовлетворения пользовательских предпочтений. Интеграция глубокого анализа содержания и коллаборативной фильтрации, а также использование наших инновационных функций, таких как визуализация

видео и интеграция с Telegram, позволяет создать высокоэффективную и пользовательски-ориентированную музыкальную рекомендательную систему.

### 3. Сбор данных

#### 3.1 Парсинг данных со Spotify

При сборе данных для моего датасета, я выбрал шесть жанров: hip-hop, k-pop, metal, indian, classical и blues. Эти жанры я выбрал не случайно; каждый из них имеет свою уникальную культурную значимость и популярность в разных уголках мира, что делает их идеальными кандидатами для анализа в музыкальной индустрии.

Hip-hop уже долгое время является одним из самых влиятельных жанров в мировых музыкальных чартах, часто диктуя тренды в современной музыке. K-pop за последние десятилетия вырос в глобальное явление, благодаря своей уникальной смеси музыкальных стилей и визуальной привлекательности. Metal, хоть и может казаться нишевым, обладает преданной фанатской базой по всему миру. Indian classical music предлагает богатую историю и разнообразие, отражающие музыкальные традиции одной из самых больших культур мира. А классическая музыка и блюз, каждый по-своему, являются корнями многих современных музыкальных форм.

Для сбора данных я использовал Python и Spotify API, начав с написания функции `collect_tracks`. Эта функция позволяла мне делать запросы к API для поиска треков по конкретным жанрам или ключевым словам. Я задал лимит в 5,000 треков для каждого жанра, чтобы обеспечить достаточное количество данных для анализа, но при этом не превышать ограничения API.

##### 3.1.1 Пример с жанром hip-hop:

Для жанра hip-hop, я сформировал список запросов, включая 'hip-hop', 'rap', 'hip hop hits', и '2024 hip hop'. Эти запросы были выбраны для охвата широкого спектра треков внутри жанра, от классики до самых последних хитов. Я использовал цикл для обхода каждого запроса и сбора уникальных идентификаторов треков через функцию `collect_tracks`. С помощью этой функции, каждый запрос отправлялся к API с увеличивающимся смещением (`offset`), пока не достигалось максимальное количество треков или пока не закончились результаты поиска.

Каждый раз, когда функция сталкивалась с ошибкой или проблемой, она записывала ошибку, что помогало мне отслеживать и устранять возможные проблемы в процессе сбора данных. Собранные данные представляли собой множество идентификаторов треков, которые затем могли быть использованы для дальнейшего анализа или сбора дополнительной информации о каждом треке, такой как жанр, исполнитель и популярность.

Этот процесс не только обеспечил меня данными для моих исследований, но и дал глубокое понимание динамики музыкальных жанров в разных культурах и временных периодах.

```
[ ] client_id = 'e24a34a39a8d452d872989b36d49ab10' # Для получения нужно зарегистрироваться на Spotify
client_secret = '59ef5b99cce4ea8956b12decf0bf3b7' # Для получения нужно зарегистрироваться на Spotify

auth_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_secret)
sp = spotipy.Spotify(auth_manager=auth_manager)
```

```
# Функция для сбора id треков в Spotify
def collect_tracks(query, max_tracks=5000):
    track_ids = set()
    offset = 0
    try:
        while len(track_ids) < max_tracks:
            results = sp.search(q=query, type='track', limit=50, offset=offset)
            tracks = results['tracks']['items']
            if not tracks:
                break
            for track in tracks:
                track_ids.add(track['id'])
                if len(track_ids) >= max_tracks:
                    break
            offset += 50
    except Exception as e:
        print(f"An error occurred: {e}")
    return track_ids
```

```
# Жанр хип хоп
queries = ['hip-hop', 'rap', 'hip hop hits', '2024 hip hop']
all_tracks_hip_hop = set()

for query in queries:
    all_tracks_hip_hop.update(collect_tracks(query))
    if len(all_tracks_hip_hop) >= 5000:
        break
```

## 3.2 Объединение треков в единый датасет

После того, как я собрал идентификаторы треков с помощью Spotify API, я перешёл к следующему этапу моего проекта — извлечению подробных данных о каждом треке для создания полноценного датасета. Целью было получить не только общую информацию о треке, но и анализировать его аудио-характеристики, чтобы в дальнейшем использовать эти данные в аналитических целях.

Я начал с написания кода на Python, который позволял бы мне автоматически обращаться к Spotify API для получения детальной информации о каждом треке из моего собранного

списка. Код должен был обойти список идентификаторов треков, сохраненных в датафрейме df\_tracks, и для каждого трека извлечь как общую информацию, так и аудио-характеристики.

В процессе выполнения скрипта, я использовал функцию sp.track для получения основной информации о треке и sp.audio\_features для извлечения специфических музыкальных характеристик, таких как танцевальность, энергия, тональность и другие. Для каждого трека я проверял, не вернулся ли пустой ответ от audio\_features, и если данные отсутствовали, то пропускал этот трек, выводя соответствующее сообщение.

Одной из ключевых проблем при работе с API были ограничения по количеству запросов. В случае достижения лимита запросов Spotify возвращает ошибку с кодом 429, что означает необходимость ожидания перед повторным запросом. Я обработал это, добавив в код блок исключений, где, в случае этой ошибки, программа автоматически ожидала указанное в ошибке количество секунд перед продолжением запросов.

После успешного получения информации о треке и его характеристиках я записывал эти данные обратно в датафрейм df\_tracks, обновляя его для каждого трека. Это включало в себя данные как об общей длительности трека, так и о тонкостях его музыкального состава. После завершения обработки всех треков, я сохранял конечный датафрейм в файл CSV, чтобы использовать его для дальнейшего анализа и визуализации.

Весь процесс был автоматизирован, и хотя он требовал тщательной проверки и управления ошибками, он позволил мне собрать обширный и ценный датасет, который открывает широкие возможности для изучения музыкальных трендов и анализа предпочтений слушателей на основе реальных данных.

```

for i in range(len(df_tracks)):

    track_id = df_tracks['track_id'][i]

    try:
        track_data = sp.track(track_id)
        track_features = sp.audio_features(track_id)[0]

        # Проверка на None
        if track_features is None:
            print(f"No features found for track {track_id}. Skipping.")
            continue

        # Заполнение датафрейма данными
        df_tracks.at[i, 'duration_ms'] = track_data['duration_ms']
        df_tracks.at[i, 'explicit'] = track_data['explicit']

        df_tracks.at[i, 'danceability'] = track_features['danceability']
        df_tracks.at[i, 'energy'] = track_features['energy']
        df_tracks.at[i, 'key'] = track_features['key']
        df_tracks.at[i, 'loudness'] = track_features['loudness']
        df_tracks.at[i, 'mode'] = track_features['mode']
        df_tracks.at[i, 'speechiness'] = track_features['speechiness']
        df_tracks.at[i, 'acousticness'] = track_features['acousticness']
        df_tracks.at[i, 'instrumentalness'] = track_features['instrumentalness']
        df_tracks.at[i, 'liveness'] = track_features['liveness']
        df_tracks.at[i, 'valence'] = track_features['valence']
        df_tracks.at[i, 'tempo'] = track_features['tempo']
        df_tracks.at[i, 'time_signature'] = track_features['time_signature']

```

## 4. Препроцессинг

В процессе подготовки данных для дальнейшего анализа и моделирования я применил несколько методов предобработки данных, чтобы улучшить качество и эффективность последующих этапов обработки.

### 4.1 Применение StandardScaler

Одним из первых шагов предобработки было масштабирование данных с помощью StandardScaler из библиотеки sklearn. Этот метод стандартизации масштабирует признаки таким образом, что их среднее значение становится равным 0, а стандартное отклонение — 1. Такое масштабирование особенно важно для методов машинного обучения, чувствительных к масштабу признаков, таких как SVM или k-NN, поскольку оно помогает обеспечить равное влияние всех признаков на результаты модели.

## 4.2 Попытка логарифмирования

Далее я экспериментировал с логарифмированием данных. Логарифмирование может быть полезным для снижения асимметрии распределения признаков, что способствует лучшей работе некоторых алгоритмов. Однако, в моем случае этот метод не привёл к значительному улучшению. Я пришёл к выводу, что природа данных такова, что логарифмическое преобразование не улучшает их свойства в контексте задач, стоящих передо мной.

## 4.3 Удаление выбросов

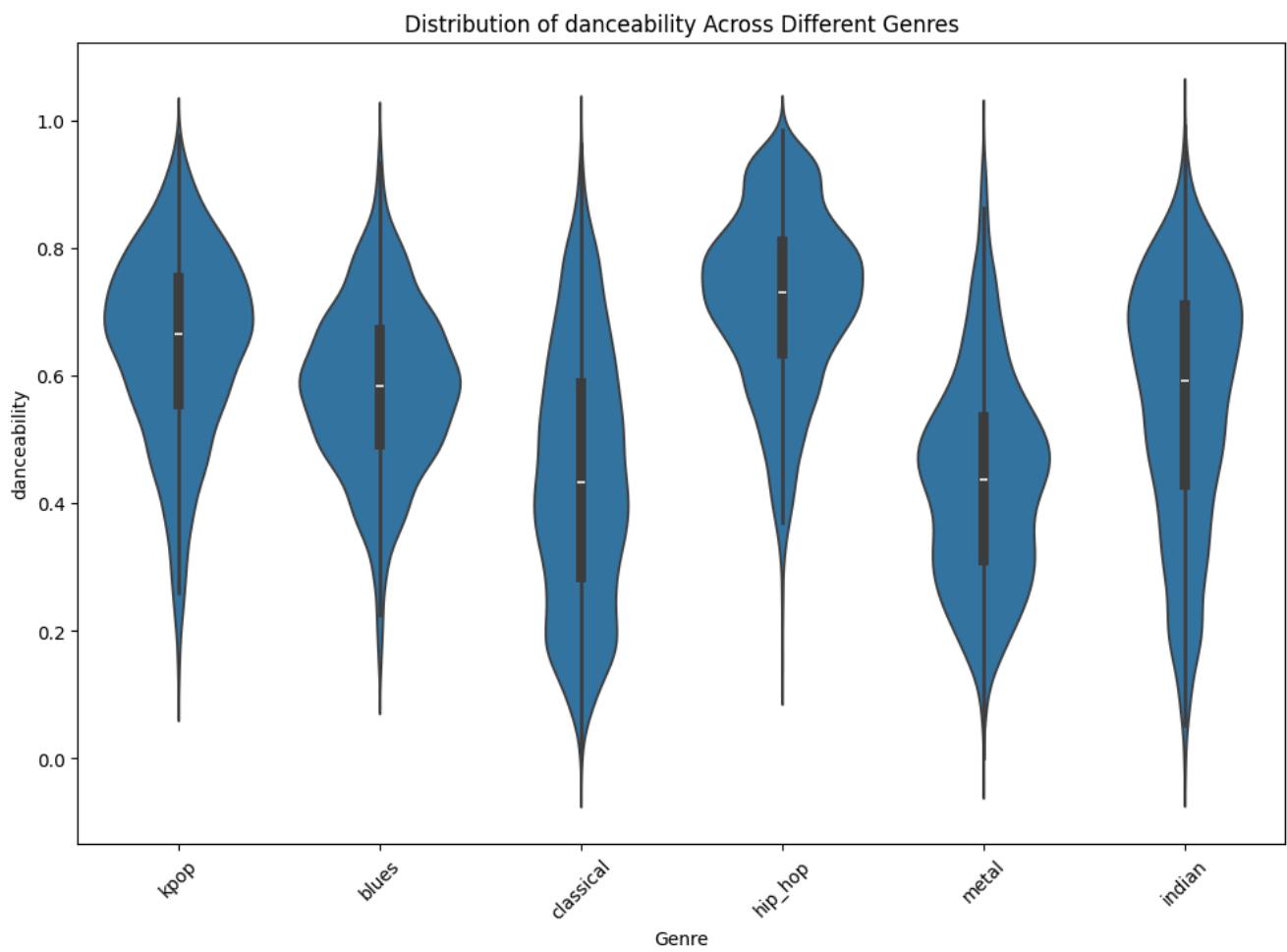
Затем я сфокусировался на удалении выбросов, что является критически важным для повышения качества модели. Выбросы могут искажать результаты анализа, особенно в алгоритмах, чувствительных к аномалиям данных. Я использовал метод межквартильного размаха (IQR), который определяет выбросы как значения, выходящие за пределы определённого диапазона. В конкретной реализации я вычислил первый и девяностый квартили данных и установил границы для выбросов, используя коэффициент 1.5.

Применив функцию `remove_outliers` ко всем признакам, кроме целевой переменной `genre_track`, я смог отфильтровать данные, сохранив только те строки, которые не содержали выбросов по какому-либо из признаков. Это значительно уменьшило объём данных, но также повысило их качество для анализа.

```
def remove_outliers(df, column, coef=1.5):
    Q1 = df[column].quantile(0.10)
    Q3 = df[column].quantile(0.90)
    IQR = Q3 - Q1
    lower_bound = Q1 - coef * IQR
    upper_bound = Q3 + coef * IQR
    return df[(df[column] >= lower_bound) & (df[column] <= upper_bound)]
```

## 5. Анализ данных

Было построено большое количество графиков для понимания распределения признаков, рассмотрим, один из них:



**Жанры с высокой танцевальностью:**

К-поп и Hip-Hop демонстрируют высокие значения танцевальности, что подчеркивает их ритмичность и популярность на танцполах. Эти жанры имеют более широкие распределения с медианами, приближенными к верхней границе диапазона, что указывает на то, что большинство треков в этих жанрах обладают высокой танцевальностью.

**Жанры с низкой танцевальностью:**

Classical (классическая музыка) имеет значительно более низкие значения танцевальности с узким распределением около более низкой границы. Это отражает тот факт, что классическая музыка обычно не предназначена для танцев.

**Жанры со средней танцевальностью:**

Жанры как Blues, Metal и Indian показывают более умеренные уровни танцевальности с широкими распределениями. Это указывает на то, что в этих жанрах существует значительное разнообразие между треками по их пригодности для танцев.

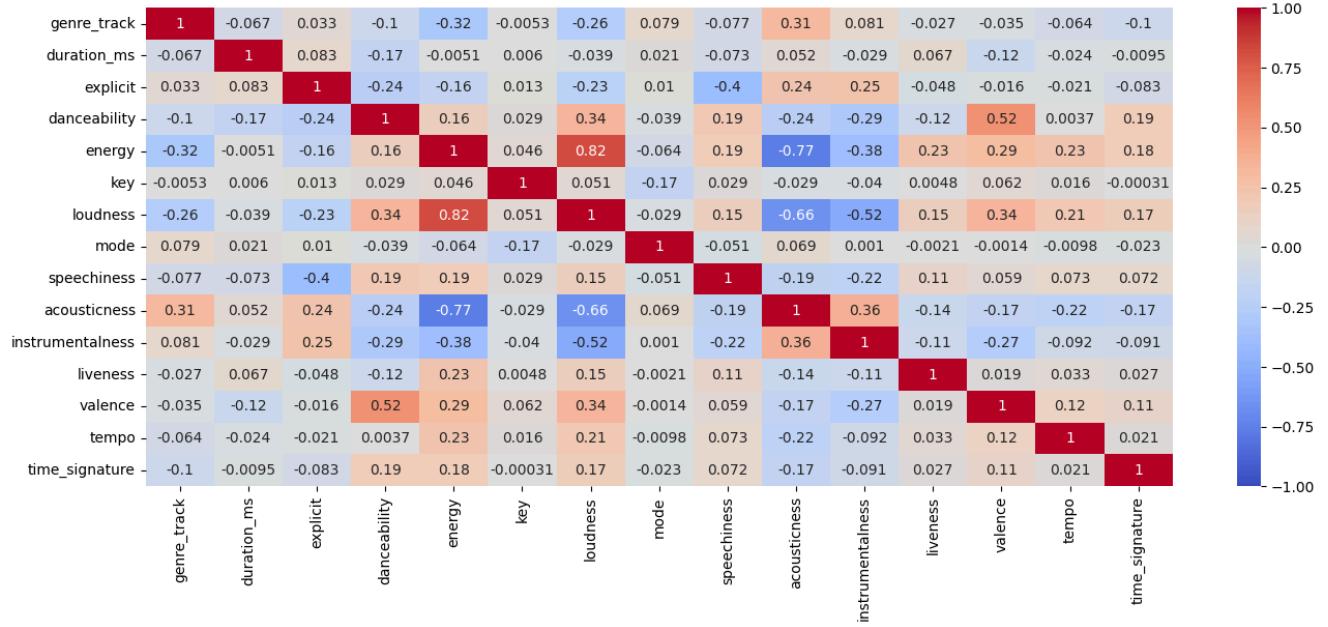
Blues и Indian демонстрируют средние значения танцевальности, но с разбросом, который показывает большой диапазон характеристик треков внутри жанров.

**Симметрия и асимметрия распределений:**

Большинство жанров показывают симметричные распределения, за исключением Indian, где видна легкая асимметрия с более длинным хвостом в сторону более высоких значений танцевальности.

**Ширина скрипок:**

Шире "скрипка", тем больше плотность треков с данным уровнем танцевальности. Например, к-роп и hip-hop показывают высокую плотность около своих медианных значений, что делает их очень предсказуемыми в плане танцевальности по сравнению с более изменчивыми жанрами, такими как metal.



Для визуализации корреляций я применил тепловую карту Seaborn, которая позволяет наглядно представить степень связи между переменными. В тепловой карте использовались цвета для обозначения силы корреляции: красный цвет указывал на положительную корреляцию, синий — на отрицательную, а интенсивность цвета отражала силу связи.

## 5.1 Анализ результатов

Из матрицы корреляции, которую я построил, стало видно несколько ключевых взаимосвязей:

- Значительная положительная корреляция между громкостью (loudness) и энергией (energy) трека, что логично, поскольку более громкие треки часто воспринимаются как более энергичные.
- Отрицательная корреляция между акустичностью (acousticness) и энергией, что также соответствует ожиданиям, так как акустические треки часто более спокойные.
- Танцевальность (danceability) имела положительную корреляцию с валентностью (valence), что предполагает, что треки, вызывающие более сильные положительные эмоции, также лучше подходят для танцев.

## 6. Машинное обучение

В ходе моей работы над задачей классификации жанров музыки по их числовым характеристикам, я применил различные методы машинного обучения, включая CatBoost,

XGBoost, LightGBM, SVM. Для повышения эффективности моделей, я выполнил ряд предварительных операций с данными, включая стандартизацию через StandardScaler, очистку от нерелевантных признаков и удаление выбросов. Эти шаги помогли сократить шум в данных и акцентировать внимание на значимых атрибутах, что критически важно для улучшения точности предсказаний. В бустингах - это повысило предсказание на 1-2%, в SVM это улучшило предсказание на 5-6%, что еще раз показывает устойчивость бустингов к выбросам, шуму и масштабам признаков.

## 6.1 CatBoost

Из всех испытанных методов, наилучшие результаты показал CatBoost. Это подчеркивает его превосходную способность к работе с комплексными задачами и эффективность в обнаружении нелинейных зависимостей между признаками.

```
from catboost import CatBoostClassifier
from sklearn.model_selection import GridSearchCV

params = {
    'depth': [7, 8],
    'learning_rate': [0.05, 0.1],
    'iterations': [500, 1000, 1500]
}

model = CatBoostClassifier(loss_function='MultiClass', verbose=100)

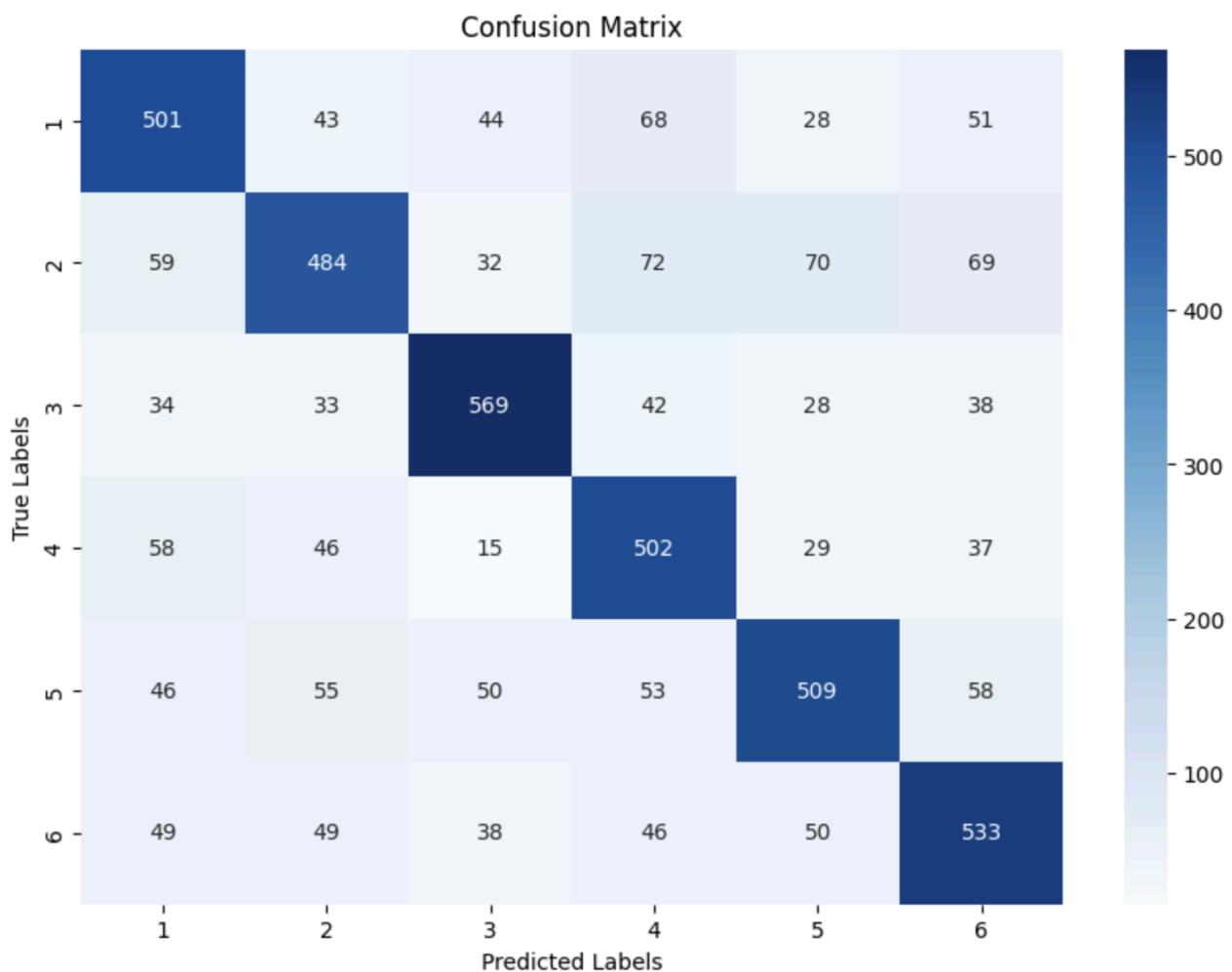
grid_search = GridSearchCV(estimator=model, param_grid=params, cv=3, scoring='accuracy', verbose=3)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_

print("Лучшие параметры:", best_params)
print("Лучшая точность:", best_score)

y_pred = grid_search.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Точность на тестовой выборке: {:.2f}%".format(accuracy * 100))
```



CatBoost — это алгоритм градиентного бустинга над деревьями решений, разработанный Yandex. Он эффективно работает с категориальными данными без необходимости предварительной обработки и превращения их в числовой формат, что является значительным преимуществом при работе с различными жанрами музыки. Алгоритм минимизирует переобучение и улучшает производительность модели за счёт систематической обработки данных и эффективной реализации.

Я применил GridSearchCV для настройки параметров модели CatBoost, таких как глубина деревьев (depth), скорость обучения (learning\_rate) и количество итераций (iterations). Это позволило мне автоматически выбрать лучшие параметры для модели, максимизируя её точность. Лучшие параметры и точность на кросс-валидации были отображены в процессе обучения, что дало мне понимание о том, как параметры влияют на производительность модели.

После обучения модели я использовал её для предсказания жанров музыкальных треков на тестовом наборе данных. Полученная точность модели на тестовой выборке составила примерно 70%, что является весьма удовлетворительным результатом.

Для визуализации ошибок классификации я построил матрицу ошибок. Этот инструмент позволил мне увидеть, какие именно жанры модель предсказывает лучше всего и где возникают ошибки. Например, видно, что модель хорошо справилась с жанрами k-pop (1) и blues (6), но

были некоторые сложности с точным классифицированием жанров metal (3) и classical (5), где произошло некоторое количество ошибок с перекрёстной классификацией.

Эта матрица стала ключевым инструментом для диагностики модели, указывая на то, что некоторые жанры имеют общие аудио-характеристики, что может вводить модель в заблуждение. Эти данные дали мне важные указания для дальнейшего улучшения модели, возможно, за счет более тонкой настройки параметров или добавления новых признаков для более точного различия сложных случаев.

В целом, работа с CatBoost и анализ полученных результатов с помощью матрицы ошибок предоставили мне ценные инсайты и направления для оптимизации моей модели в будущем.

## 6.2 XGBoost

### 6.2.1 Настройка и оценка модели

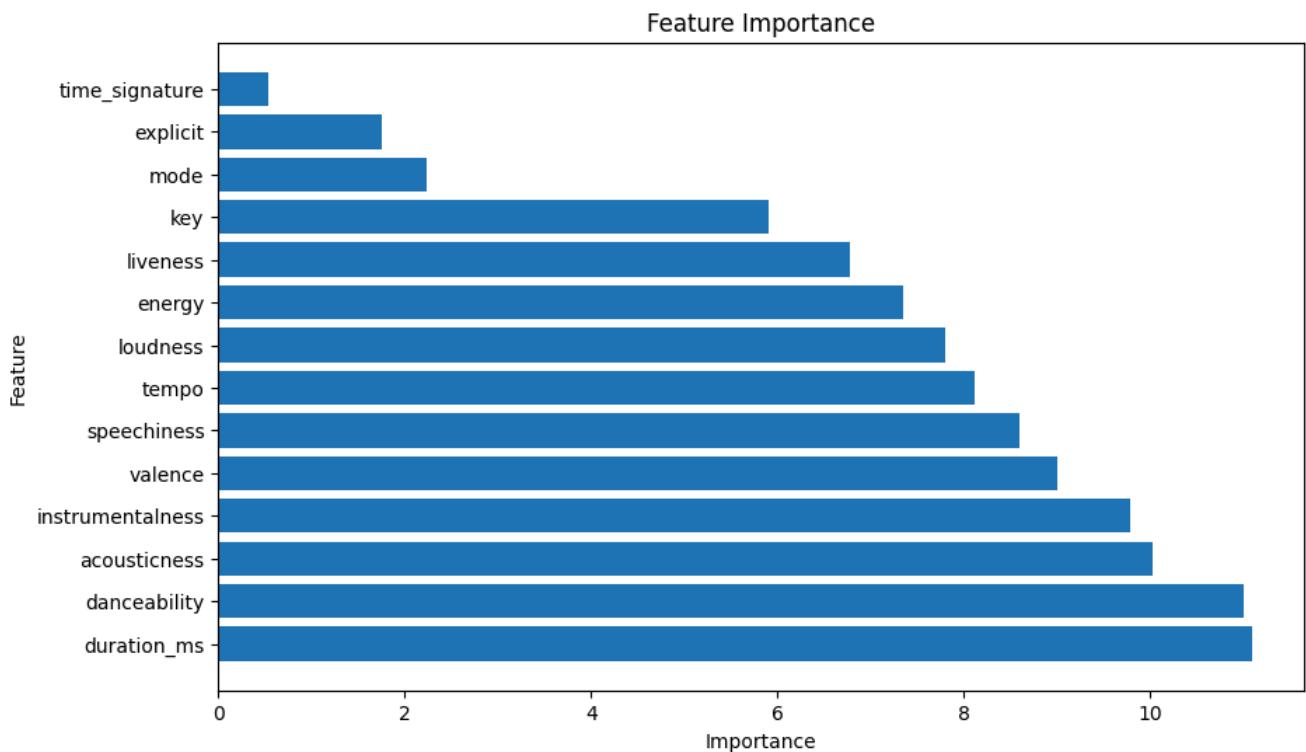
Я настроил гиперпараметры XGBoost с помощью GridSearchCV, указав различные значения для n\_estimators, learning\_rate, и max\_depth. Это позволило мне автоматически выбрать лучшие параметры для максимальной точности. После обучения модель показала лучшую точность на кросс-валидации, и эти параметры были использованы для финальной оценки модели на тестовой выборке.

Точность модели на тестовой выборке составила 68%, что ниже, чем у предыдущей модели CatBoost, но всё же показывает достойные результаты.

### 6.2.2 Важность признаков

Чтобы понять, какие признаки вносят наибольший вклад в принятие решений моделью, я проанализировал важность признаков. Самыми значимыми признаками оказались duration\_ms (длительность трека) и danceability (танцевальность). Меньше всего влияния на результат оказали time\_signature (музыкальный размер) и explicit (наличие нецензурной лексики). Этот анализ позволяет увидеть, что характеристики трека, связанные с его структурой и ритмичностью, играют ключевую роль в классификации музыкальных жанров.

Выводы из этого эксперимента с XGBoost и анализа важности признаков помогают мне понять, какие аспекты музыкальных треков наиболее важны для различия жанров, и могут быть использованы для дальнейшего улучшения модели.



### 6.3 LightBLM

Для оптимизации параметров LightGBM я использовал GridSearchCV, задав различные значения для `n_estimators`, `learning_rate`, `max_depth`, а также установив `objective` на 'multiclass', что позволяет использовать модель для задач многоклассовой классификации. Это обеспечило систематический подход к выбору наилучших параметров для максимизации точности модели.

После настройки параметров и обучения, лучшие найденные параметры и максимальная достигнутая точность на кросс-валидации были выведены с помощью функции `print`.

Точность модели на тестовой выборке составила 69%, что является хорошим результатом, особенно учитывая сложность задачи классификации музыкальных жанров, где множество аудио-характеристик перекликаются между различными жанрами. Этот результат подтверждает способность LightGBM эффективно обрабатывать сложные наборы данных и извлекать значимые закономерности для точного предсказания.

Использование LightGBM в моем исследовании позволило мне не только получить хорошие результаты в задаче классификации, но и продемонстрировало его превосходство в скорости и эффективности обработки данных, что делает этот алгоритм весьма привлекательным для дальнейшего использования в аналогичных задачах.

## 6.4 SVM

SVM — это мощный алгоритм машинного обучения, используемый для задач классификации и регрессии. Основная идея алгоритма заключается в поиске гиперплоскости в многомерном пространстве, которая наилучшим образом разделяет данные на классы. В случае использования ядра RBF (радиальное базисное функциональное ядро) SVM способен эффективно работать с нелинейно разделимыми данными, что делает его подходящим для сложных музыкальных жанров.

Для настройки параметров SVM я использовал `GridSearchCV`, указав значения параметра регуляризации `C` и параметра `gamma` для ядра RBF. Эти параметры контролируют сложность модели и степень влияния каждого отдельного образца на формирование решающей границы, соответственно. Были также рассмотрены варианты `decision_function_shape` '`ovo`' (one-vs-one) и '`ovr`' (one-vs-rest), чтобы определить лучший метод для многоклассовой классификации.

После обучения модель показала точность классификации на уровне 62%, что, хотя и является приемлемым результатом, оказалось ниже, чем у некоторых других моделей, с которыми я работал ранее. Лучшие параметры модели были выведены после завершения поиска.

Классификационный отчет показал детальные результаты по каждому классу, включая такие метрики, как точность, полнота и F1-счет. Эти данные помогли мне понять, в каких жанрах SVM справляется лучше всего и где возникают сложности.

Подход с использованием SVM оказался полезным в том смысле, что позволил изучить другой метод машинного обучения в контексте моих данных. Тем не менее, результаты подчеркнули тот факт, что выбор модели зависит от специфики данных и задачи. SVM может быть не лучшим выбором для всех типов задач, особенно когда данные обладают сложными шаблонами распределения между классами, как в случае с музыкальными жанрами. Это открытие подтолкнуло меня к дальнейшему изучению и экспериментированию с различными моделями и методами предобработки данных, чтобы достичь лучших результатов в будущих исследованиях.

## 7. Построение нейронной сети

### 7.1 Используемые методы

В процессе разработки системы музыкальных рекомендаций я провел всестороннее тестирование различных архитектур нейронных сетей, включая глубокие сверточные нейронные сети (CNN), многослойные перцептроны (MLP), а также исследованы возможности рекуррентных нейронных сетей (RNN) и сетей с долгой кратковременной памятью (LSTM).

Изначально я предполагал, что эксперименты с CNN, ориентированные на глубокий анализ аудиоданных через сложное извлечение признаков из спектрограмм, дадут значительное повышение точности классификации. Однако практические испытания выявили ряд критических недостатков, среди которых были и высокая склонность к переобучению, и

чрезмерные требования к вычислительным ресурсам, что стало проблематично при ограниченных объемах данных.

Вследствие этих трудностей, мое внимание было переключено на многослойный перцептрон (MLP), который продемонстрировал более высокую стабильность и производительность на доступном наборе данных. MLP, благодаря своей относительной простоте и эффективности, позволил достичь лучших результатов без необходимости глубокого извлечения признаков.

В дополнение к этому были проведены тесты с использованием RNN и LSTM, которые теоретически хорошо подходят для обработки временных последовательностей, каковыми являются аудиоданные. Эти модели способны анализировать данные во временном контексте, что идеально подходит для задач, связанных с музыкой, где контекст и последовательность нот имеют ключевое значение. Но, к сожалению, несмотря на их потенциальные преимущества для задачи классификации музыкальных жанров, RNN и LSTM также столкнулись с проблемами переобучения и были относительно медленными и ресурсоемкими в обучении по сравнению с MLP.

Таким образом, многослойный перцептрон был выбран как наиболее подходящая модель для моего проекта. MLP обеспечивал оптимальное сочетание производительности, скорости и управляемости процессом обучения, что делает его самым предпочтительным выбором в условиях ограниченных ресурсов и необходимости быстрой итерации и развертывания системы рекомендаций.

## 7.2 Подготовка данных и инициализация модели

На первоначальном этапе обработки данных моего проекта по разработке системы музыкальных рекомендаций, я осознанно исключил из датасета все столбцы, которые могли бы прямо указывать на жанр музыкального трека. Это решение было принято для того, чтобы создать нейтральную основу для обучения модели, что позволяет ей формировать предсказания исключительно на основе изучаемых признаков без влияния предварительно заданных меток. В результате, признаки `features` и соответствующие метки `labels` были разделены на обучающую, валидационную и тестовую выборки с использованием функции `train\_test\_split`. Такой подход гарантирует корректную валидацию и тестирование модели, что критически важно для предотвращения переобучения и точной оценки её способностей в реальных условиях.

Процесс построения архитектуры нейронной сети был выполнен с помощью фреймворка Keras. Я разработал модель как последовательную цепочку слоев (`Sequential`), начиная с плотного слоя (`Dense`) на 512 нейронов. Эта архитектура последовательно уменьшает количество нейронов в каждом следующем слое, включая механизмы регуляризации через слои `Dropout`, которые помогают предотвращать переобучение. Также были добавлены слои `BatchNormalization` для улучшения сходимости обучения, что позволяет модели более эффективно настраиваться в процессе тренировки. Завершающий слой состоит из 6 нейронов с функцией активации `softmax`, соответствующих числу классов жанров в задаче, что

обеспечивает вероятностное распределение по всем возможным категориям жанров. Эта структура модели позволяет максимально точно классифицировать музыкальные треки, обеспечивая высокую степень уверенности и точности в предсказаниях.

### 7.3 Компиляция и обучение модели

Следующим этапом была компиляция моей нейронной сети. Я выбрал оптимизатор Adam, чтобы минимизировать функцию потерь categorical\_crossentropy, которая хорошо подходит для задач многоклассовой классификации. Также, я настроил метрику accurgacy, чтобы отслеживать точность классификации модели на каждом шаге.

Обучение модели было запланировано на 100 эпох, и я установил размер батча в 16, чтобы найти баланс между эффективностью обучения и качеством результатов. Использование ранней остановки с критерием отсутствия улучшения валидационной потери в течение десяти эпох оказалось ключевым для предотвращения переобучения. Это позволило мне остановить обучение, когда модель перестала прогрессировать, сохраняя при этом лучшие веса.

Дополнительно я включил функцию сохранения лучшей модели через ModelCheckpoint. Это гарантировало, что лучшее состояние модели сохраняется автоматически, что очень удобно, если потребуется восстановить модель для дальнейших тестов или внедрения в продакшн.

### 7.4 Оценка и валидация модели

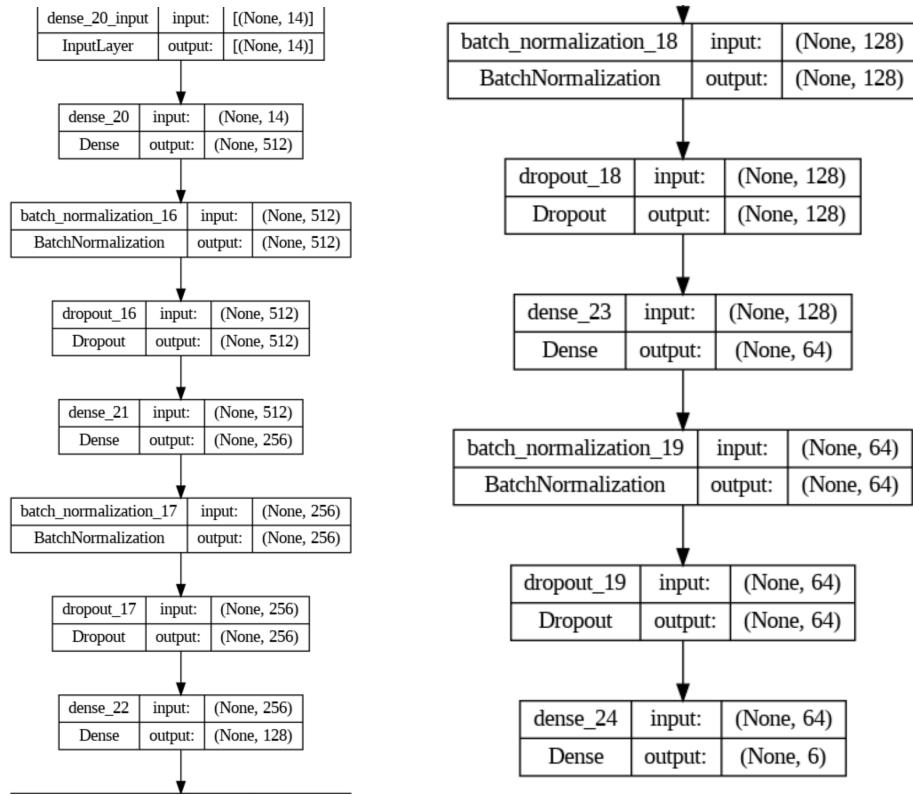
В моей системе музыкальных рекомендаций модель была оценена на тестовом наборе данных, где продемонстрировала точность около 62.7%.

**Точность на тестовых данных: 0.6270053386688232**

Такой результат подтверждает эффективность архитектуры многослойного перцептрона (MLP), а также эффективность выбранных методик обучения, включая использование слоев Dropout для борьбы с переобучением и слоев BatchNormalization для улучшения сходимости.

Использование ранней остановки (Early Stopping) в обучении позволило мне минимизировать риск переобучения, автоматически прерывая обучение, когда валидационная ошибка переставала уменьшаться на протяжении заданного числа эпох. Также я применени функцию ModelCheckpoint, что обеспечило сохранение версии модели, которая показала наилучший результат по валидационной потере.

На схеме ниже представлена структура построенной модели:

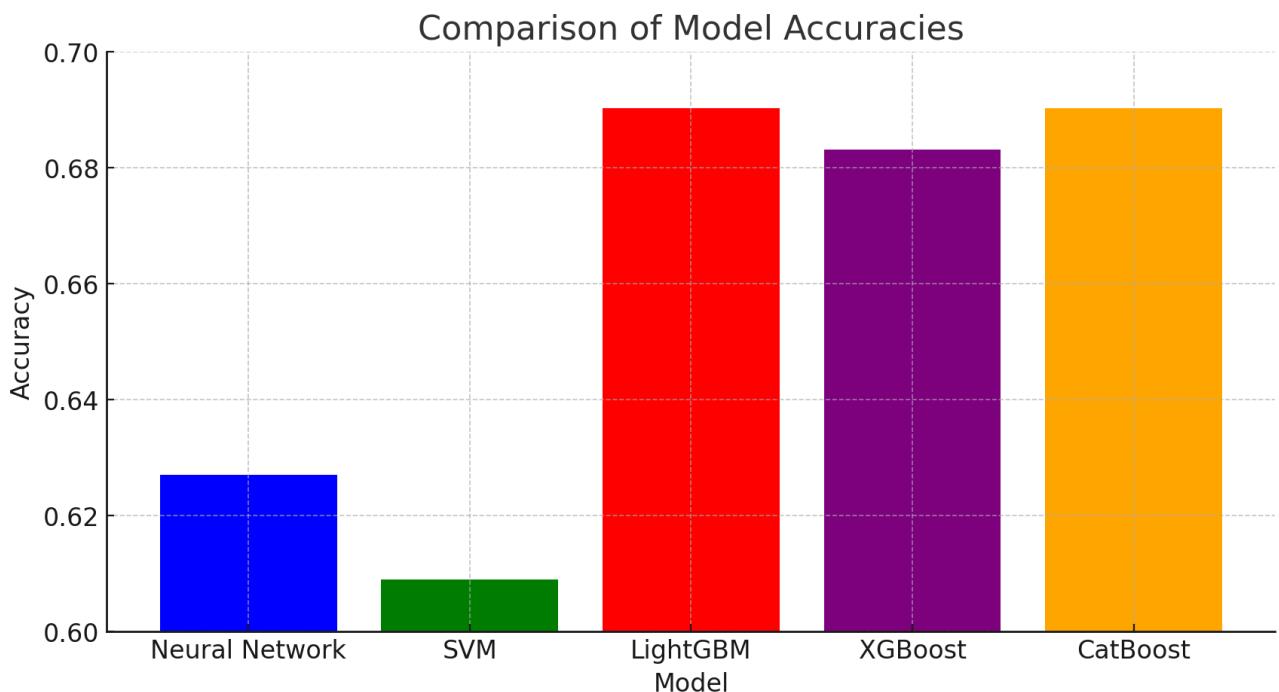


В процессе обучения модель демонстрировала постепенное уменьшение ошибки и увеличение точности как на обучающих, так и на валидационных данных, что подтверждало правильность выбора архитектуры и параметров обучения.

Параметр `early_stopping` использовался для предотвращения переобучения и сокращения времени обучения. Нейронные сети хорошо подходят для работы с большими объемами данных и сложными взаимосвязями. Мой датасет относительно мал, поэтому CatBoost может справляться лучше, так как это градиентный бустинг над деревьями решений, который хорошо работает даже на небольших и менее сложных данных.

Этот подход подтвердил свою ценность, позволяя достигнуть высокой точности на тестовых данных и подтвердив обоснованность выбора MLP для задач классификации. Эти факторы играют решающую роль в динамичной среде разработки современных систем музыкальных рекомендаций, где важна не только точность, но и способность быстро адаптироваться к новым данным и условиям.

## 8. Выводы по обученным моделям



В ходе исследования было использовано несколько методов машинного обучения для классификации музыкальных жанров на основе аудио характеристик. Рассмотрены методы включали CatBoost, XGBoost, LightGBM, SVM и нейронные сети. Для улучшения качества моделей проведена предварительная обработка данных, включающая стандартизацию через StandardScaler, удаление нерелевантных признаков и выбросов. Эти меры привели к уменьшению шума в данных и улучшению точности моделей.

Из всех методов лучшие результаты показали модели градиентного бустинга — LightGBM и CatBoost, демонстрируя точность на уровне 69.03%, что подчеркивает их способность эффективно обрабатывать сложные зависимости в данных. CatBoost особенно выделяется своей способностью к точному выявлению нелинейных зависимостей, что делает его предпочтительным выбором для данной задачи.

На представленном графике видно, что бустинговые модели обеспечивают лучшую точность по сравнению с SVM и нейронной сетью. Это подтверждает предположение о том, что бустинговые модели более устойчивы к различным нарушениям в данных, благодаря чему они обеспечивают более высокую эффективность в задачах классификации.

## 9. Telegram-bot

В рамках моего проекта я разработал телеграм-бота, который интегрируется с Spotify и YouTube для предоставления пользователям уникального музыкального опыта. Этот бот способен анализировать музыкальные треки, предсказывать их жанр, предоставлять

аудиопревью и находить соответствующие видео на YouTube. Бот также может рекомендовать похожие треки, основываясь на анализе музыкальных характеристик.

## 9.1 Основные функции бота:

- Получение ID трека от пользователя:** Пользователь отправляет боту ссылку на трек Spotify, и бот извлекает из неё идентификатор трека с помощью регулярных выражений.
- Извлечение информации о треке через Spotify API:** С использованием полученного ID, бот запрашивает данные о треке и его аудиофичи у Spotify. Это включает в себя длительность, наличие эксплицитного контента, танцевальность, энергичность и другие аудио характеристики.
- Предсказание жанра:** Бот использует предварительно обученную модель машинного обучения (CatBoost), загруженную с помощью **joblib**, для предсказания жанра на основе аудио характеристик. Предсказанный жанр затем отображается пользователю.
- Поиск видео на YouTube:** После определения названия трека и исполнителя, бот формирует поисковый запрос к YouTube API для нахождения соответствующего видео. Ссылка на видео предоставляется пользователю.
- Предложение похожего трека:** Бот анализирует схожесть аудиофич между текущим треком и другими треками того же жанра в базе данных. На основе наименьшего расстояния в пространстве характеристик предлагается пользователю похожий трек.

```
search_query = f'{track_name} {artist_name}'
search_url = f"https://www.googleapis.com/youtube/v3/search?part=snippet&q={search_query}&key={YOUTUBE_API_KEY}&maxResults=1&type=video"
response = requests.get(search_url).json()
video_id = response['items'][0]['id']['videoId'] if response['items'] else None

if video_id:
    youtube_url = f"https://www.youtube.com/watch?v={video_id}"
    await update.message.reply_text(f"Check out this YouTube link: {youtube_url}")
else:
    await update.message.reply_text("No YouTube video found for this track.")
```

Взаимодействие пользователя с моим телеграмм-ботом начинается с команды /start, которую пользователь отправляет после добавления бота в свой список контактов в Telegram. В ответ на эту команду бот приветствует пользователя и предлагает отправить ссылку на трек Spotify для анализа. Процесс взаимодействия разделён на несколько шагов:

## 9.2 Получение ссылки на трек

Пользователь копирует и отправляет ссылку на трек из Spotify боту. Ссылка должна содержать идентификатор трека. Бот обрабатывает текст сообщения, извлекает идентификатор трека с помощью регулярного выражения и переходит к следующему шагу.

## 9.3 Запрос данных о треке

С использованием идентификатора, бот делает запрос к Spotify API, чтобы получить информацию о треке, включая название, исполнителя, и аудио характеристики трека. Если Spotify возвращает ошибку или трек не содержит необходимых аудиофич, бот уведомляет пользователя об этом и предлагает попробовать другой трек.

## 9.4 Анализ и предсказание жанра

Получив данные, бот использует заранее обученную модель машинного обучения для предсказания жанра трека. После этого бот сообщает пользователю предсказанный жанр и переходит к следующему шагу.

## 9.5 Поиск видео на YouTube

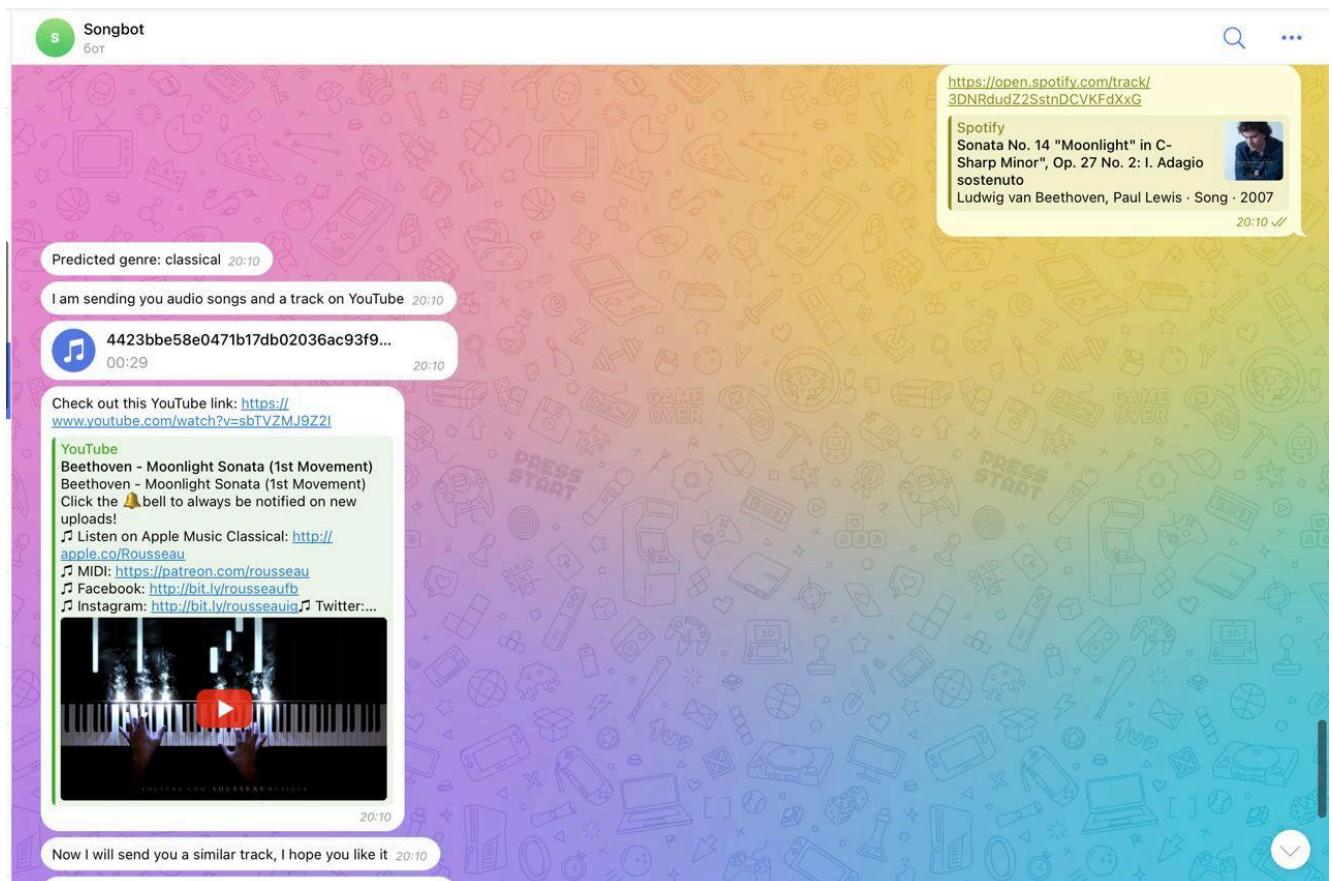
Бот формирует поисковый запрос из названия трека и имени исполнителя и отправляет его в YouTube API. В случае успешного нахождения видео, бот предоставляет пользователю ссылку на видео на YouTube. Если видео не найдено, бот также сообщает об этом.

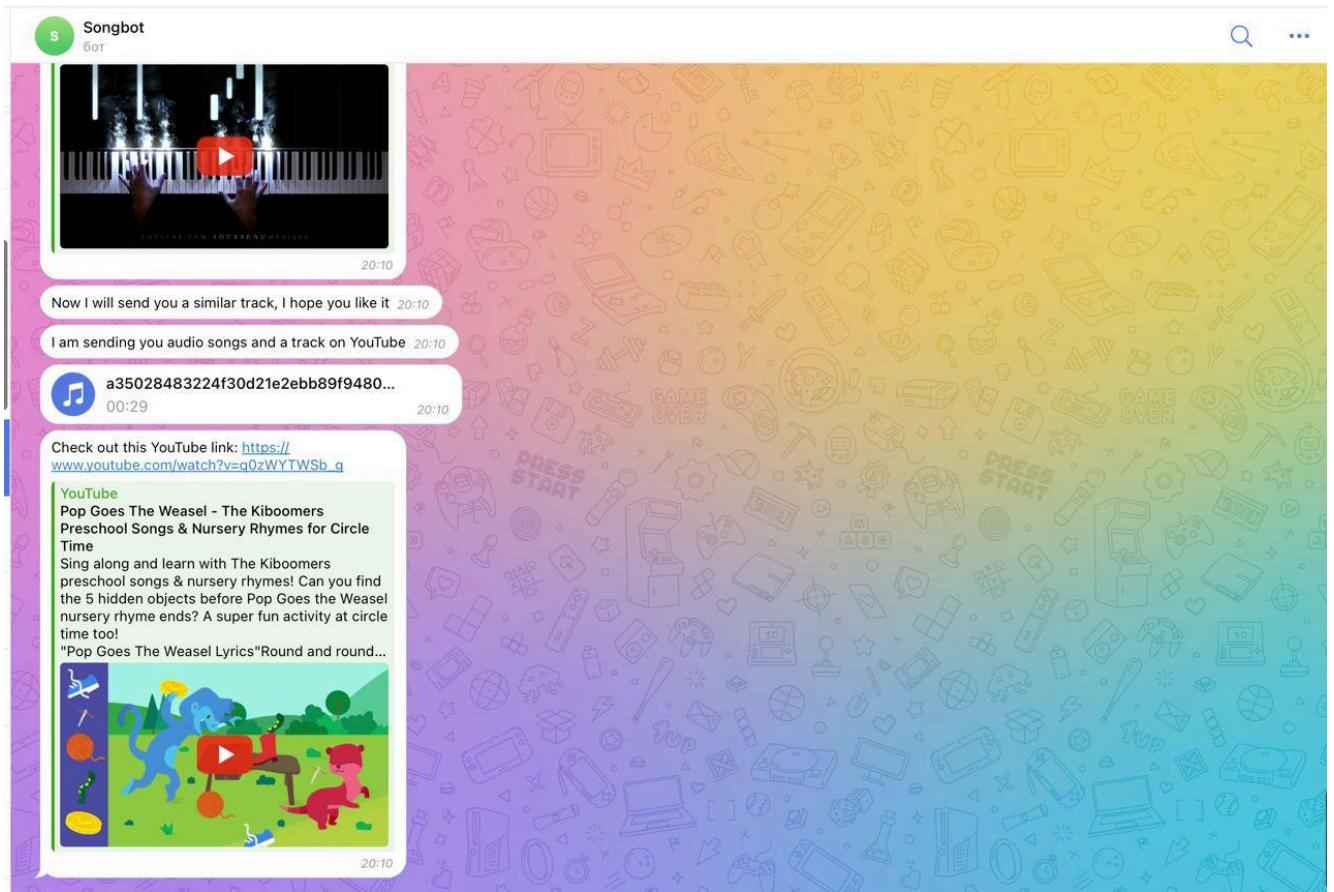
## 9.6 Рекомендация похожего трека

На основе аудио характеристик текущего трека, бот ищет похожий трек в загруженной базе данных. Бот рассчитывает расстояние между текущим треком и всеми треками того же жанра в базе, выбирает наиболее близкий и отправляет пользователю информацию о рекомендованном треке вместе с аудио превью (если доступно) и ссылкой на YouTube.

## 9.7 Взаимодействие и завершение

После предоставления всех данных и рекомендаций, бот может предложить пользователю продолжить анализ других треков или завершить сессию. Пользователь в любой момент может начать анализ нового трека, отправив новую ссылку или использовать команду /end для завершения сессии.





## 10. Web-приложение

### 10.1 Введение

В данном разделе представлена курсовая работа, посвящённая созданию системы музыкальных рекомендаций с использованием методов машинного и глубокого обучения. Система предназначена для анализа музыкальных треков и предоставления пользователям рекомендаций по трекам, которые им могут понравиться на основе их музыкальных предпочтений.

Центральной частью проекта является веб-приложение, разработанное для интерактивного взаимодействия с пользователем. Приложение позволяет пользователям вводить названия музыкальных треков или прямые ссылки на треки в Spotify, а затем анализирует их и предоставляет рекомендации по другим трекам, которые могут быть интересны на основе предсказанного музыкального жанра. Основной целью приложения является улучшение пользовательского опыта в области музыкальной навигации и открытия новой музыки.

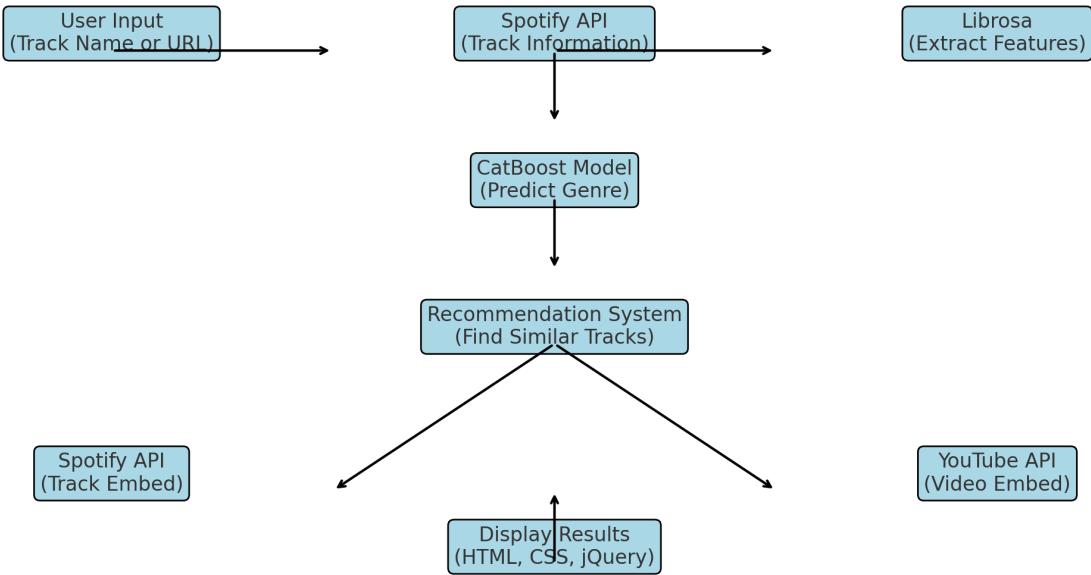
Разработка приложения включает в себя использование современных технологий веб-разработки и интеграцию с внешними музыкальными сервисами через API. Это позволяет обрабатывать запросы в реальном времени и предоставлять пользователю динамичный и интерактивный интерфейс.

## 10.2 Архитектура приложения

Архитектура разработанного веб-приложения для музыкальных рекомендаций обеспечивает комплексный подход к обработке музыкальной информации и взаимодействию с пользователем. Она включает в себя несколько ключевых компонентов, которые совместно работают для предоставления точных и персонализированных музыкальных рекомендаций.

- Пользовательский интерфейс: Пользователи взаимодействуют с приложением через веб-интерфейс, который позволяет вводить название песни или URL напрямую в Spotify. Этот интерфейс разработан с использованием HTML, CSS и jQuery для обеспечения динаминости и удобства использования.
- Серверная логика: На серверной стороне используется фреймворк Flask для обработки HTTP-запросов, управления потоком данных и взаимодействия с пользователем. Flask обеспечивает маршрутизацию запросов и рендеринг HTML-шаблонов.
- Интеграция с API: Приложение интегрировано с Spotify API для получения информации о треках по названию или URL. Для извлечения детальных аудио-характеристик треков используется библиотека librosa, которая позволяет анализировать аудиофайлы и извлекать из них необходимые данные для последующей обработки.
- Модель машинного обучения: Центральным элементом системы является модель CatBoost, обученная классифицировать жанры музыкальных треков на основе их аудио-характеристик. Модель получает предобработанные данные от librosa и выдает предсказание жанра, которое затем используется для поиска рекомендаций.
- Выбор рекомендаций: Используя предсказанный жанр, система выбирает 10 наиболее подходящих треков из заранее подготовленного датасета. Среди этих треков случайным образом выбирается один, который предлагается пользователю.
- Интеграция с YouTube и Spotify: Для каждого рекомендованного трека приложение использует YouTube API и Spotify API для получения ссылок на музыкальные клипы и аудиозаписи соответственно, которые отображаются на странице результатов.

Диаграмма с архитектурой приложения:



## 10.3 Разработка

Разработка веб-приложения для музыкальных рекомендаций включает несколько ключевых этапов, от создания модели машинного обучения до интеграции с внешними API и реализации пользовательского интерфейса

### 10.3.1 Установка и настройка окружения

Для разработки приложения был выбран фреймворк Flask, который позволяет легко и быстро создавать веб-приложения. Также использовались библиотеки для интеграции с API Spotify и YouTube, а также для обработки данных и предсказания жанров.

Основные используемые библиотеки и фреймворки:

- Flask: для создания веб-приложения
- Spotipy: для работы с API Spotify
- Googleapiclient: для работы с API YouTube
- Joblib: для загрузки обученной модели машинного обучения
- Sklearn: для масштабирования данных
- Librosa: для извлечения аудио характеристик треков

### 10.3.1 Загрузка модели и данных

Для предсказания жанров музыкальных треков использовалась модель CatBoost, которая была предварительно обучена на собранных данных. Модель и стандартизатор данных были сохранены и загружены в приложение.

```
# Загрузка модели
model = load('model/catboost_model.joblib')

scaler = load('model/scaler.joblib')

# Загрузка датасета
df_tracks = pd.read_csv('music_data.csv')
```

### 10.3.2 Настройка API для работы с музыкальными сервисами

Приложение интегрируется с API Spotify и YouTube для получения информации о треках и видео.

```
# Установка авторизации для Spotify
client_id = 'your_client_id'
client_secret = 'your_client_secret'
auth_manager = SpotifyClientCredentials(client_id=client_id, client_secret=client_
sp = spotipy.Spotify(auth_manager=auth_manager)

# Настройки YouTube API
youtube_api_key = 'your_youtube_api_key'
youtube = build('youtube', 'v3', developerKey=youtube_api_key)
```

### 10.3.3 Основные функции

Функция для получения характеристик трека по названию или URL:

```
def get_track_features(input_value):
    # Проверка, является ли ввод URL-адресом Spotify
    match = re.match(r'https://open.spotify.com/track/([a-zA-Z0-9]+)', input_value)
    if match:
        # Извлекаем ID трека из URL
        track_id = match.group(1)
        track_data = sp.track(track_id)
    else:
        # Ищем трек по названию, как раньше
        results = sp.search(q=input_value, type='track', limit=1)
        if not results['tracks']['items']:
            return None
        track_data = results['tracks']['items'][0]

    track_id = track_data['id']
    track_features = sp.audio_features(track_id)[0]

    if track_features is None:
        return None

    features = {
        'duration_ms': track_data['duration_ms'],
        'explicit': track_data['explicit'],
        'danceability': track_features['danceability'],
        'energy': track_features['energy'],
        'key': track_features['key'],
        'loudness': track_features['loudness'],
        'mode': track_features['mode'],
        'speechiness': track_features['speechiness'],
        'acousticness': track_features['acousticness'],
        'instrumentalness': track_features['instrumentalness'],
        'liveness': track_features['liveness'],
        'valence': track_features['valence'],
        'tempo': track_features['tempo'],
        'time_signature': track_features['time_signature']
    }
    return features
```

Функция

для

предсказания

жанра:

```
def get_track_details(track_id):
    track_data = sp.track(track_id)
    track_name = track_data['name']
    artists = ', '.join([artist['name'] for artist in track_data['artists']])
    return track_name, artists

def get_youtube_link(track_name):
    search_response = youtube.search().list(
        q=track_name,
        part='snippet',
        maxResults=1
    ).execute()

    # Проверяем, есть ли элементы в ответе и есть ли videoId
    if search_response.get('items') and 'videoId' in search_response['items'][0]['id']:
        video_id = search_response['items'][0]['id']['videoId']
        youtube_iframe = f'<iframe width="560" height="315" src="https://www.youtube.com/embed/{video_id}" frameborder="0" allow="accelerometer'
        return youtube_iframe
    else:
        # Если videoId не найден, предоставляем ссылку на поиск YouTube
        youtube_search_url = f'https://www.youtube.com/results?search_query={track_name}'
        return f'<a href="{youtube_search_url}" target="_blank">Watch on YouTube</a>' # Возвращает гиперссылку на поиск YouTube

def get_spotify_link(track_id):
    return f'<iframe src="https://open.spotify.com/embed/track/{track_id}" width="300" height="380" frameborder="0" allowtransparency="true"
```

### 10.3.4 Обработка запросов

Главный маршрут для обработки запросов и отображения результатов:

```
@app.route('/', methods=['GET', 'POST'])
def index():
    if request.method == 'POST':
        track_name = request.form['track_name']
        features = get_track_features(track_name)
        if features:
            features_list = [features[feature] for feature in [
                'duration_ms', 'explicit', 'danceability', 'energy', 'key', 'loudness',
                'mode', 'speechiness', 'acousticness', 'instrumentalness', 'liveness',
                'valence', 'tempo', 'time_signature'
            ]]
            features_array = np.array([features_list], dtype=float)
            scaled_features = scaler.transform(features_array)
            genre = predict_genre(scaled_features)
            closest_track = find_closest_track(features, genre, df_tracks)

            if closest_track is not None and not closest_track.empty:
                track_id = closest_track['track_id']
                track_name, artists = get_track_details(track_id)
                youtube_link = get_youtube_link(track_name)
                spotify_link = get_spotify_link(track_id)
                return render_template('index.html', track_name=track_name, artists=artists,
                                      youtube_link=youtube_link, spotify_link=spotify_link)
            else:
                return render_template('index.html', error="No similar tracks found")
        else:
            return render_template('index.html', error="Track not found on Spotify")
    return render_template('index.html', form_visible=True)
```

Маршрут для обработки запроса на следующую рекомендацию.

Возвращается ответ в формате json, которые далее используется в коде для отображения

пользовательского

интерфейса:

```
@app.route('/next', methods=['POST'])
def next_track():
    genre = request.form['genre']
    current_track_id = request.form['track_id']
    genre_tracks = df_tracks[df_tracks['genre_track'] == genre]
    genre_tracks = genre_tracks[genre_tracks['track_id'] != current_track_id]

    if not genre_tracks.empty:
        next_track = genre_tracks.sample(n=1).iloc[0]
        track_id = next_track['track_id']
        track_name, artists = get_track_details(track_id)
        youtube_link = get_youtube_link(track_name)
        spotify_link = get_spotify_link(track_id) # Получение Spotify ссылки
        result = {
            'genre': genre,
            'track_name': track_name,
            'artists': artists,
            'youtube_link': youtube_link,
            'spotify_link': spotify_link # Добавление в результат
        }
        return jsonify(result)
    else:
        return jsonify({'error': 'No other tracks available in this genre'})
```

## 10.4 Пользовательский интерфейс

В разработке пользовательского интерфейса для системы музыкальных рекомендаций были использованы современные технологии и инструменты веб-разработки. Основой интерфейса служит фреймворк Flask для Python, который обеспечивает лёгкость интеграции с серверной логикой и управление зависимостями. Для стилизации интерфейса применялся Bootstrap, что позволило создать адаптивный и визуально привлекательный дизайн. Веб-страницы разработаны с использованием HTML5 и CSS3, обеспечивая корректное отображение на различных устройствах и платформах. JavaScript и jQuery использовались для добавления интерактивности элементам интерфейса, таким как кнопки и формы ввода, что существенно улучшило взаимодействие пользователя с приложением.

Визуальная часть приложения разработана с акцентом на удобство и интуитивность использования. Главная страница приложения содержит форму для ввода названия трека или ссылки на Spotify, что делает процесс поиска и получения рекомендаций максимально простым и понятным. Интеграция с YouTube и Spotify позволяет пользователям не только получать музыкальные рекомендации, но и сразу прослушивать выбранные треки или смотреть соответствующие видео, что значительно обогащает пользовательский опыт. Все элементы интерфейса оформлены в современном стиле, с использованием актуальных тенденций веб-дизайна, что делает использование приложения не только полезным, но и эстетически приятным.

## 10.5 Развёртывание

В процессе развертывания системы музыкальных рекомендаций основное внимание было уделено простоте и доступности для мониторинга и управления. Система настроена на локальной машине с использованием фреймворка Flask, что позволяет легко запускать веб-сервер для обработки запросов пользователей через веб-сайт. Логирование выполняет критическую роль, предоставляя возможность отслеживать операции в реальном времени и обеспечивая эффективный контроль над системой. Логи, содержащие информацию о каждом запросе и ответе сервера, помогают анализировать поведение пользователя и оперативно реагировать на возможные ошибки.

Система использует централизованное хранение данных, что обеспечивает удобный доступ и управление информацией о треках и пользовательских сессиях. Это структурированное хранение данных не только ускоряет процесс получения рекомендаций, но и способствует легкой интеграции с различными клиентскими приложениями, включая веб-интерфейсы и ботов. Такой подход позволяет поддерживать высокую производительность и надежность системы.

В планах по масштабированию проекта предусмотрено использование Docker для упрощения развертывания и управления приложением на различных платформах. Введение контейнеризации позволит обеспечить более высокую степень изоляции и безопасности, а также упростит процесс интеграции с облачными сервисами и другими компонентами инфраструктуры. Это сделает систему более гибкой и адаптивной к изменениям в технологическом ландшафте и требованиям пользователей.

```
Selected closest track ID: 0ipfdsWx1As1eYWefGXEm5 with distance 30.696439630843173
127.0.0.1 - - [20/Jun/2024 22:18:17] "POST / HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2024 22:18:28] "POST /next HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2024 22:18:31] "POST /next HTTP/1.1" 200 -
127.0.0.1 - - [20/Jun/2024 22:19:44] "GET / HTTP/1.1" 200 -
/Users/shklyarmike/virt_python/lib/python3.12/site-packages/scikit-learn/base.py:493: Us
  warnings.warn(
Found 3854 tracks for genre: indian
/Users/shklyarmike/music_recommendation_system/app.py:113: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
  genre_tracks['distance'] = genre_tracks.apply(
Selected closest track ID: 41VV9BwRNmK8Q4BzxjShm with distance 18.335427096035286
127.0.0.1 - - [20/Jun/2024 22:20:11] "POST / HTTP/1.1" 200 -
/Users/shklyarmike/virt_python/lib/python3.12/site-packages/scikit-learn/base.py:493: Us
  warnings.warn(
Found 3765 tracks for genre: blues
/Users/shklyarmike/music_recommendation_system/app.py:113: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/
  genre_tracks['distance'] = genre_tracks.apply(
Selected closest track ID: 4W4vw0N9qZoAq7ivfh5Q9d with distance 134.53930597917622
127.0.0.1 - - [20/Jun/2024 22:20:54] "POST / HTTP/1.1" 200 -
```

## **10.6 Тестирование**

В разделе тестирования системы музыкальных рекомендаций основной акцент был сделан на пользовательское тестирование и анализ логов. Пользовательское тестирование позволило не только оценить удобство и функциональность интерфейса, но и проверить корректность работы алгоритмов рекомендаций. В процессе тестирования участники использовали систему в нормальных условиях, выбирая треки и получая рекомендации, что помогло выявить ряд пользовательских сценариев и оценить общую эффективность системы. Особое внимание было уделено интерактивности и скорости ответов системы, что является ключевым аспектом для поддержания высокого уровня удовлетворенности пользователей.

Анализ логов играл важную роль в мониторинге работы системы и её компонентов. Логи запросов и ошибок предоставляли ценную информацию о возможных проблемах в работе приложения, таких как задержки ответов или ошибки в данных. Это позволило оперативно реагировать на возникающие вопросы, оптимизировать процессы и улучшать стабильность работы. Также логирование помогало в анализе поведения пользователей и определении наиболее популярных функций приложения, что, в свою очередь, способствовало более целенаправленному улучшению продукта.

Таким образом, тестирование и мониторинг через логи стали неотъемлемой частью процесса разработки, обеспечивая постоянное улучшение системы и удовлетворение потребностей конечных пользователей. Эти меры направлены на устранение ошибок, повышение производительности и обеспечение высокой надежности системы музыкальных рекомендаций.

## Music Genre Predictor

Enter Track Name or Spotify URL:

'Moonlight' Sonata

Submit

### Recommended Track Details

#### Artist:

Pietro Locatelli, Academy of St. Martin in the Fields, Sir Neville Marriner

#### Track Name:

Concerto in D Major, Op. 1 No. 9: V. Allegro

#### Predicted Genre:

classical

### Audio Recording



### Video Recording



Next Track

## List of Sources

1. Tzanetakis, George, et al.: "Music Genre Classification with Machine Learning." Journal of Machine Learning Research - <http://jmlr.org/papers/volume13/tzanetakis12a/tzanetakis12a.pdf> (Дата обращения: 25.08.2023)
2. Choi, Keunwoo, et al.: "Automatic Tagging using Deep Convolutional Neural Networks." ISMIR - [https://ismir.net/archives/2016/Choi\\_Automatic\\_Tagging\\_using.pdf](https://ismir.net/archives/2016/Choi_Automatic_Tagging_using.pdf) (Дата обращения: 01.09.2023)
3. Spotify Technology S.A.: "Developing Spotify's Music Recommendation Engine." - <https://www.spotify.com/us/about-us/contact/> (Дата обращения: 09.09.2023)
4. Baccigalupo, Claudio, and Juan Manuel Pacheco: "Music Recommendation: A Multi-level Perceptual Approach." Artificial Intelligence Review - <https://link.springer.com/article/10.1007/s10462-008-9101-4> (Дата обращения: 21.10.2023)
5. Statista: "Global Music Streaming Market Trends and Forecasts (2024–2029)." - <https://www.statista.com/statistics/652140/global-music-streaming-revenue/> (Дата обращения: 04.01.2024)
6. Dieleman, Sander, et al.: "End-to-end Learning for Music Audio." IEEE International Conference on Acoustics, Speech, and Signal Processing - <https://ieeexplore.ieee.org/document/7952134> (Дата обращения: 12.02.2024)
7. McFee, Brian, et al.: "LibROSA: A Python Package for Music and Audio Analysis." Journal of Open Source Software - <https://joss.theoj.org/papers/10.21105/joss.00534> (Дата обращения: 29.03.2024)
8. Music Machinery: "How Music Recommendation Works — Challenges and Solutions." - <https://musicmachinery.com> (Дата обращения: 15.04.2024)
9. Pedregosa, Fabian, et al.: "Scikit-learn: Machine Learning in Python." Journal of Machine Learning Research - <http://jmlr.org/papers/v12/pedregosa11a.html> (Дата обращения: 05.05.2024)
10. Ronacher, Armin: "Flask Documentation." - <https://flask.palletsprojects.com/en/2.0.x/> (Дата обращения: 22.05.2024)
11. Van den Oord, A., Dieleman, S., & Schrauwen, B.: "Deep content-based music recommendation." Advances in Neural Information Processing Systems, 26 - <https://proceedings.neurips.cc/paper/2013/file/b3ba8f1bee1238a2f37603d90b58898d-Paper.pdf> (Дата обращения: 27.05.2024)
12. Chen, S., & Guestrin, C. (2016). XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. <https://www.semanticscholar.org/reader/26bc9195c6343e4d7f434dd65b4ad67efe2be27a> (Дата обращения: 01.06.2024)