Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра электронных вычислительных машин

К ЗАЩИТЕ ДОПУСТИТЬ:
Зав. каф. ЭВМ
_____ Д.И. Самаль

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к дипломному проекту
на тему
МОДУЛЬ ВЫДЕЛЕНИЯ ИНФОРМАЦИОННЫХ ОБРАЗОВ ИЗ МУЗЫКАЛЬНОГО ПРОИЗВЕДЕНИЯ

БГУИР ДП 1-40 02 01 01 004 ПЗ

Студент П.С. Арабей

Руководитель Н.Н. Иванов

Консультанты:

от кафедры ЭВМ Н.Н. Иванов

по экономической части И.В. Смирнов

Нормоконтролёр А.С. Сидорович

Рецензент

МИНСК 2017

СОДЕРЖАНИЕ

РЕФЕРАТ

Дипломный проект предоставлен следующим образом. Электронные носители: 1 компакт-диск. Чертежный материал: 6 листов формата А1. Пояснительная записка: 81 страниц, 27 рисунков, 6 таблиц, 18 литературных источников, 4 приложения.

Ключевые слова: информационные признаки, жанровая кассификация, API, визуализация данных.

Объектом исследования и разработки является возможность выделять информационные образы из музыкального произведения для сервиса рекомендации музыки и жанровой классификации.

Целью данного дипломного проекта является создание программного модуля, который выделял информационные образы из музыкального трека только на основании акустического анализа.

При разработке программного средства использовалась среда разработки PyCharm, pip, консольная утилита Lame, библиотека scipy, библиотека scikit-learn, библиотека matplotlib, язык программирования Python.

Областью практического применения программного средства является сервис рекомендации музыки. Разрабатываемый программный продукт подразумевает собой набор фундаментальных классов и модулей, реализующих взаимодействие с файловой системой и базой данных. Данные компоненты интегрируются с логикой получения информационных образов.

Разработанный программный продукт можно считать экономически эффективным, и он полностью оправдывает вложенные в него средства.

Дипломный проект является завершенным, поставленная задача решена в полной мере, присутствует возможность эффективного расширения программного продукта путем реализации новых компонентов.

ВВЕДЕНИЕ

На сегодняшний день количество музыкальный треков в Интернете не поддаётся подсчёту. В одном только сервисе Яндекс.Музыка доступно 17 миллионов треков [1] . Сегодня выпускается тысячи песен каждый день [2]. Существует сотни жанров и поджанров музыки. Поэтому задача поиска новой музыки становится нетривиальной.

Для решение данной задачи используются рекомендательные сервисы, которые используют четыре подхода к анализу музыки для составления рекомендации:

1. Популярность трека. Использование данных о популярности трека и информации о пользователях. Этот способ доступен только большим онлайн сервисам. Качество рекомендации зависит от количества пользователей данного сервиса. Популярность треков определяется количеством и качеством отзывов и количеством покупок.

2. Метаинформация о треке. Это способ также доступен большим онлайн сервисам с большой библиотекой музыки. Качество рекомендации зависит от размеров библиотеки. Метаинформация о треке представляет собой тег с жанром, информация об исполнителе, альбом в который включён этот трек и т.д.

3. Семантика текста. Анализируя музыкальные блоги с применением технологий обработки естественных языков подход позволяет постоянно изучать веб и аналитически просматривать десятки миллионов страниц, имеющих отношения к музыке. Текст песни анализируется на основе взвешенных лексем. Произведения рекомендуется при близости его дескрипторов с дескрипторами пользователя.

4. Акустико-синтаксический анализ. На текущий момент нет методов достоверного распознавания музыкальных инструментов и музыкальных звуков. Однако, несмотря на это анализ сигнала играет очень важную роль используется в работе рекомендательных алгоритмов. Анализируются фрагменты произведения длительности от 200 мс до 4 с, в зависимости от вариабельности мелодии. Для каждого сегментов произведения определяется громкость, тембр, выявляются наиболее громко звучащие музыкальные инструменты; устанавливается к какой части композиции (припев, куплет и т. д.) относится этот сегмент.

В первом способе информационные признаки о треке получается на основе анализа поведения пользователя. Рекомендации при использовании данного метода делаются на основе коллаборативной фильтрации на основе соседей. Во втором методе является метаинформация о треке, а рекомендация делается на основе коллаборативной фильтрации на основе модели. В современных мультимедийных сервисах(youtube, lastfm, Яндекс.Музыка) используется гибридный подход, которых объединяет в себе подход осно-

ванный на соседстве и основанный на модели [3] [4].

Целью данного дипломного проекта является создание модуля, который выделял информационные образы из музыкального трека только на основании акустического анализа. Информационный образ - это признаковое описание трека как музыкального сигнала. Такой сервис должен быть лишён субъективности и обеспечить качество рекомендации.

В соответствии с поставленной целью были определены следующие задачи:

- выделение спектральных, временных и иных признаков из музыкального трека;
- проверка значимости признаков путём использовании их в задаче жанровой классификации(задача настройки классификатора решаться не будет);
- визуализация данных алгоритмом t-SNE.

1 ОБЗОР ЛИТЕРАТУРЫ

В части рассмотренных исследований решались смежные задачи, такие как жанровая классификация, идентификация музыкального трека, нахождение заимствований в музыкальном треке, распознавание звуковых источников. Для решения данных задач использовались разные подходы к выделения информационных образов из музыкальных треков. Среди рассмотренных приложений также стоит отметить разнообразие подходов к выделению информационных признаков. В данной дипломной работе стоит задача найти такие характерные информационные образы, которые позволили на основе их делать рекомендацию

1.1 Обзор методов выделения фрагментов из музыкальных произведений

В работе[5] информационные признаки, которые будут рассмотрены в следующем подразделе, рассчитываются по фрагментам музыкального трека в 1 секунду, состоящему из 40 непересекающихся окон в 25 миллисекунд. Для классификации используется 30 фрагментов. Полная сборка аудиоданных используемая в данной работе состоит из 15 жанров * 50 файлов * 30 секунд = 22500 секунд (то есть 6,25 часов аудио). Каждый фрагмент анализируется независимо. По каждому окну получают набор числовых? признаков. Их математическое ожидание и дисперсию используют в качестве признаков для фрагмента. В работе[6] используется такой же подход. Трёхсекундный фрагмент состоит из 1500 окон по 0,02 секунды. В отличии от работ[5] по автоматической жанровой классификации музыкального трека, в качестве признаков фрагмента используют среднее значение, дисперсию, коэффициент асимметрии, коэффициент эксцесса информационных признаков окон. Эти четыре значения затем используются для измерения подобия. В работе[7], где описывается принцип работы большинства аудиоидентификационных систем, используются фрагменты музыкального трека размером от 10 до 500 миллисекунд с перекрытием фрагментов от 50 до 90 процентов. В этой же работе описываются алгоритмы распознавания ремиксов.

Каждый музыкальный трек проходит передискретизацию до частоты 44100 Гц. Затем берется четырёхсекундный фрагмент.

Недостатками данных методов является потеря информации о структурах в музыкальном треке, которые не помещаются в один фрагмент.

1.2 Обзор методов представления музыкального трека в спектрально-временном виде

В работе[8] решается задача идентификации(опознавания) музыкального трека, а также поиск заимствований в жанре музыки хип-хоп. Для

получения спектрограммы используется оконное преобразования Фурье с окном Хемминга и размером окна 64 миллисекунды. После получения спектрограммы используется только амплитудная составляющая с последующим логарифмированием и применением высокочастотного фильтра к спектральной кривой. Эти шаги предпринимаются для того, чтобы сделать спектрограмму менее зависимой от абсолютного уровня и грубой спектральной формы. Также в работе рассмотрено constant Q преобразование[9]. Выделение информационных признаков в работе состоит из трёх стадий. Конечным результатом является коррелограмма. Первая стадия моделирует влияние звуковой волны на базилярную мембрану ушной улитки с помощью полосового фильтра, который реализуется четырьмя каскадными секциями фильтра второго порядка, которые реализуют фильтр восьмого порядка с импульсной характеристикой «gammatone» с центральной частотой в 9,3 кГц. Вторая стадия моделирует движение волосковых клеток. Так как клетки реагируют только на отклонение ресничек в одном направлении, и это вводит этап выпрямления в цепочке обработки сигналов. На низких частотах волосковые клетки имеют тенденцию срабатывать в определенной фазе сигнала - процесс, называемый блокировкой фазы. По мере того как частота входного сигнала увеличивается, блокировка фазы начинает заканчиваться примерно на 1,5 кГц и исчезает на 5 кГц, но при отсутствии блокировки тонкой структуры формы сигнала волосковые клетки фиксируются в амплитудной огибающей сигнала. Этот эффект моделируется посредством операции свертки с импульсным откликом фильтра типа «приподнятого-косинуса» [10] с T = 0,25 мс. Третья стадия это построение коррелограммы. Коррелограмма моделирует человеческое восприятия основного тона. Для этого используется автокорреляция. Измерением средней энергии в зависимости от задержки можно определить базовый период сигнала. После FFT-преобразования считается оконная автокорреляция для частотной области. Полученный трёхмерный массив данных(время, частота, высота тона) используется для получения информационных признаков для классификации музыкальных инструментов. В работе симулируется человеческое восприятие. Механизмы распознавания в человеческом мозге гораздо лучше приспособлены к сложным акустическим средам, чем любой искусственный механизм (рисунок 1.1).

В работе для извлечения информационных образов о ритме используется вейвлет Добеши четвёртого порядка.

1.3 Обзор методов выделения информационных признаков

В работе различается два вида признаков: спектральные и временные. Временные признаки - это значения, которые вычисляются из значений самих аудиоданных. Спектральные - на основе спектра полученного преобразованием Фурье.

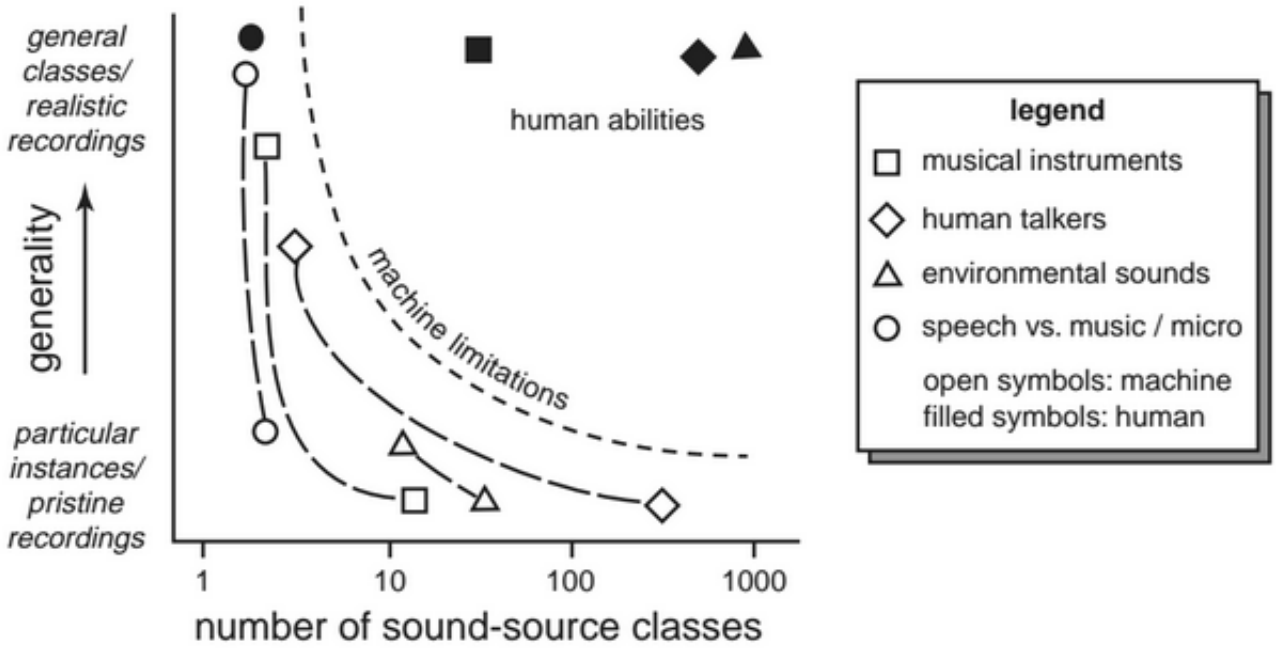

Рисунок 1.1 – График показывающий ограничения искусственных механизмов распознавания

Временные признаки:

1. Количество переходов сигнала через ноль.

$$ZeroCrossignRate = \frac{1}{N_S} \sum_{i=0}^{N_S-2} sgn(s_i) \oplus sgn(s_{i+1}) \tag{1.1}$$

где N_S - количество сэмплов в окне, $sgn()$ - функция которая возвращает 1 при положительном значении аргумента, и 0 при отрицательном. s - вектор, содержащий данные сигнала для одного окна.

2. Автокорреляция первого порядка.

$$FirstOrderAutocorrelation = \frac{1}{N_S} \sum_{i=0}^{N_S-2} s_i * s_{i+1} \tag{1.2}$$

где N_S - количество сэмплов в окне, s - вектор, содержащий данные сигнала для одного окна.

3. Энергия сигнала.

$$Energy = \frac{1}{N_S} \sum_{i=0}^{N_S-1} (s_i)^2 \tag{1.3}$$

где N_S - количество сэмплов в окне, s - вектор, содержащий данные сигнала

для одного окна.

Спектральные признаки:

1. Линейная регрессия спектра

$$\beta = \frac{N_B \sum_{i=1}^{N_B}(a_i F_i) - \sum_{i=1}^{N_B} a_i \sum_{i=1}^{N_B} F_i}{N_B \sum_{i=1}^{N_B}(F_i)^2 - \sum_{i=1}^{N_B}(F_i)^2} \tag{1.4}$$

где a - амплитуда, N_B - количество компонент разложения, а F - частота соответствующей амплитуды .

Это статистическое приближение использовано для нахождения наклона (β) спектра, который является мерой отношения высокочастотной составляющей к низкочастотной составляющей звука и, следовательно, тембра.

2. Среднее арифметическое взвешенное спектра.

$$SpectralCentroid = \frac{\sum_{i=1}^{N_B} a_i F_i}{\sum_{i=1}^{N_B} a_i} \tag{1.5}$$

Психоакустически это показатель воспринимаемой «яркости» звука, обеспечивающий лучшую оценку «яркого» звука, нежели тональность.

3. Гладкость спектра, как мера спектральной огибающей.

$$SSmoothness = \sum_{i=1}^{N_B} 20 \log a_i - \frac{20 \log a_{i-1} + 20 \log a_i + 20 \log a_{i+1}}{3} \tag{1.6}$$

где a - амплитуда, а N_B - количество компонент разложения. Белый шум, спектр которого имеет энергию на всех частотах будет иметь гладкость 1. Синусоидальный сигнал имеет только один пик в спектре и гладкость спектра будет равняться 0.

4. Дисперсия спектра относительно его среднего взвешенного. Оркестровые треки будут иметь большее значение, в отличии от сольных или монофонический треков. Вычисляется по следующей формуле:

$$SSpread = \sqrt{\frac{\sum_{i=1}^{N_B} a_i (F_i - SpectralCentroid)^2}{\sum_{i=1}^{N_B} a_i}} \tag{1.7}$$

где a - амплитуда, $SpectralCentroid$ - центроида спектра, N_B - количество компонент разложения, а F - частота соответствующей амплитуды .

5. Коэффициент асимметрии сигнала, является мерой того, насколько искажён спектр относительно его среднего взвешенного.

$$SDissymmetry = \sqrt{\frac{\sum_{i=1}^{N_B} a_i(F_i - SpectralCentroid)^3}{\sum_{i=1}^{N_B} a_i}} \tag{1.8}$$

В работе[5] исследованы алгоритмы автоматической жанровой классификации. Предложены набор признаков для представления «музыкальных аспектов» и ритмических структур аудиосигнала.

Термин «музыкальные аспекты» используются для обозначения особенности музыки, связанные с текстурой, тембром и музыкальными инструментами. В работе используется 4 статистических информационных образов, представляющие собой мат. ожидания и СКО следующих числовых признаков:

1. Среднее арифметическое взвешенное спектра, как мера спектральной яркости (см. формулу 1.5).

2. Энергетические спектральные окна по уровню 0,85.

$$\sum_{i=1}^{Rolloff} a_i = \sum_{i=1}^{N_B} a_i \tag{1.9}$$

где a - амплитуда, а N_B - количество компонент разложения.

3. Производная по частотам.

$$Flux = \|a_i - a_p\| \tag{1.10}$$

где a_i - амплитуда текущего окна, a_p - амплитуда прошлого окна.

4. Количества переходов через ноль сигнала, как показатель зашумленности сигнала (см. формулу 1.1).

Дополнительно выделяется процент окон, чья спектральная энергия меньше чем средняя спектральная энергия по фрагменту.

Также выделяются ритмические признаки. Вычисление признаков для представления ритмической структуры музыки основано на вейвлет преобразовании с вейвлетом Добеши. К результату вейвлет преобразования применяется автокорреляционная функция. Фиксируется первые пять пиков автокорреляционной функции и их соответствующие периодичности в ударах в минуту рассчитываются и добавляются в «ритмическую» гистограмму (см. рисунок 1.2). Этот процесс повторяется итерацией по сигналу и накоплением периодичности в гистограмме.

Пики гистограммы соответствуют различным периодичности звукового сигнала и используются в качестве основы для расчета признаков рит-

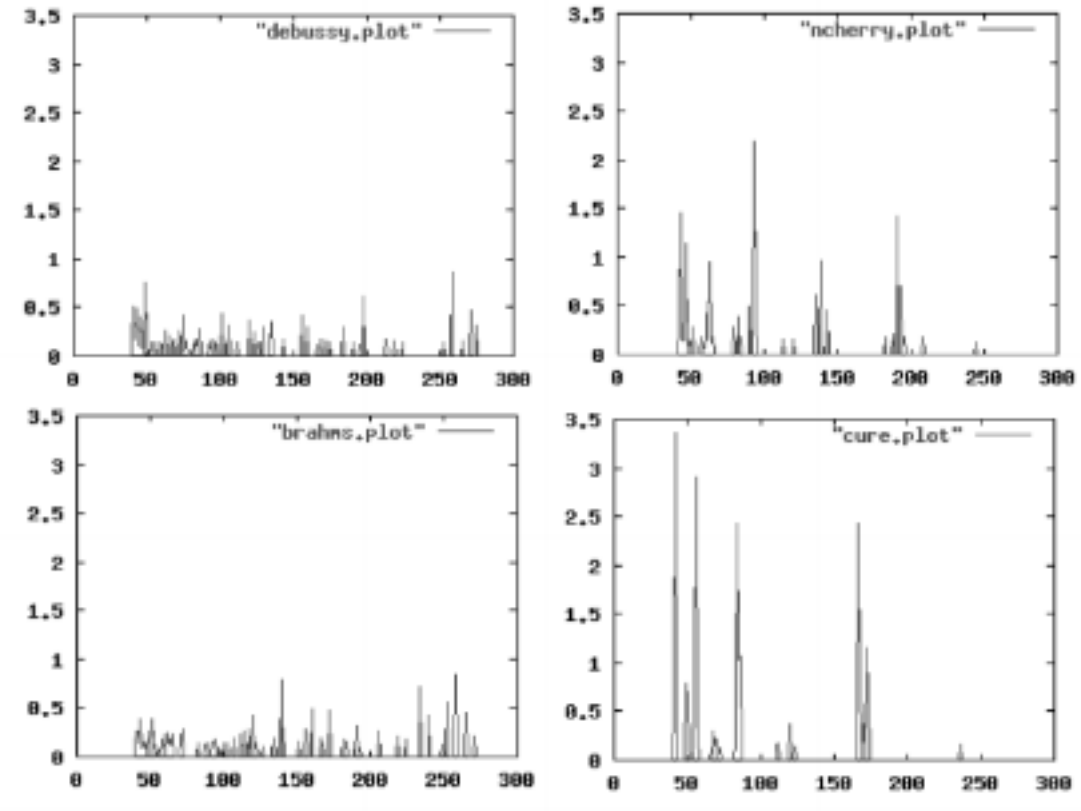

Рисунок 1.2 – Гистограмма ритма. Слева классическая музыка, справа поп-музыка

ма. Используются следующие признаки, основанные на «ритмической» гистограмме :

1. Период - 0: Периодичность в ударах в минуту первого пика.
2. Амплитуда - 0: Относительная амплитуда (деленная на сумму амплитуд) первого пика.
3. Отношение периодичности - 1: отношение периодичности второго пика к периодичности первого пика.
4. Амплитуда - 1: Относительная амплитуда второго пика.
5. Отношение периодичности - 2: отношения периодичности третьего пика к периодичности второго пика.
6. Амплитуда - 2: Относительная амплитуда третьего пика.
7. Отношение периодичности - 3: отношения периодичности четвёртого пика к периодичности третьего пика.
8. Амплитуда - 3: Относительная амплитуда третьего пика.

8-мерный вектор признаков, используемый для представления ритмической структуры и силы, комбинируется с 9-мерным вектором музыкальной поверхности(4 статистических информационных образа * 2 параметра), чтобы сформировать 17-мерный вектор признаков, который использу-

ется для автоматической классификации музыкального жанра. В работе [6] вводят меру спектральной плоскостности и коэффициент амплитуды.

$$SFM_k = \frac{[\prod S_k^2(f)]^{\frac{1}{N}}}{\frac{1}{N}\sum_f S_k^2(f)} \tag{1.11}$$

$$SCF_k = \frac{max_k S_K^2(k)}{\frac{1}{N}\sum_k S_K^2(k)} \tag{1.12}$$

где S^2 - спектральная плотность мощности.

Чувственный эквивалент этих признаков можно охарактеризовать как шумоподобность и тоноподобие. Признаки с чувственным значением представляют характеристики звука, которые с большей вероятностью сохраняются в течении произведения и, следовательно, должны быть более надежными

1.4 Мел-кепстральные коэффициенты(MFCC)

Мел - психофизическая единица высоты звука, применяется главным образом в музыкальной акустике. Количественная оценка звука по высоте основана на статистической обработке большого числа данных о субъективном восприятии высоты звуковых тонов. Звуковые колебания частотой 1000 Гц при эффективном звуковом давлении $2 * 10^{-3}$ Па (то есть при уровне громкости 40 фон), воздействующие спереди на наблюдателя с нормальным слухом, вызывают у него восприятие высоты звука, оцениваемое по определению в 1000 мел. Звук частоты 20 Гц при уровне громкости 40 фон обладает по определению нулевой высотой (0 мел). Зависимость нелинейна, особенно при низких частотах (для «низких» звуков). Преобразовать значение частоты звука (Гц) в значение высоты (мел) можно по формуле:

$$m = 1127{,}01048 \ln(1 - \frac{f}{700}) \tag{1.13}$$

При обработке звука мел-частотный кепстр (MFC) представляет собой кратковременный спектр мощности звука, основанный на косинус - преобразовании Фурье на логарифмической спектральной мощности на нелинейной мел-шкале частот.

Мел-кепстральные коэффициенты (MFCC) - это коэффициенты, которые в совокупности образуют MFC. Различие между кепстром и мел-кепстром в том, что в MFC полосы частот равномерно распределены по шкале мела, что приближается к восприятию слуховой системы человека более близко, чем линейные интервалы частот, используемые в нормальном кепстре.

MFCC обычно используются в качестве образов в системах распознавания речи, таких как системы, которые могут автоматически распознавать номера телефонов, произносимые в телефоне. MFCC также все чаще на-

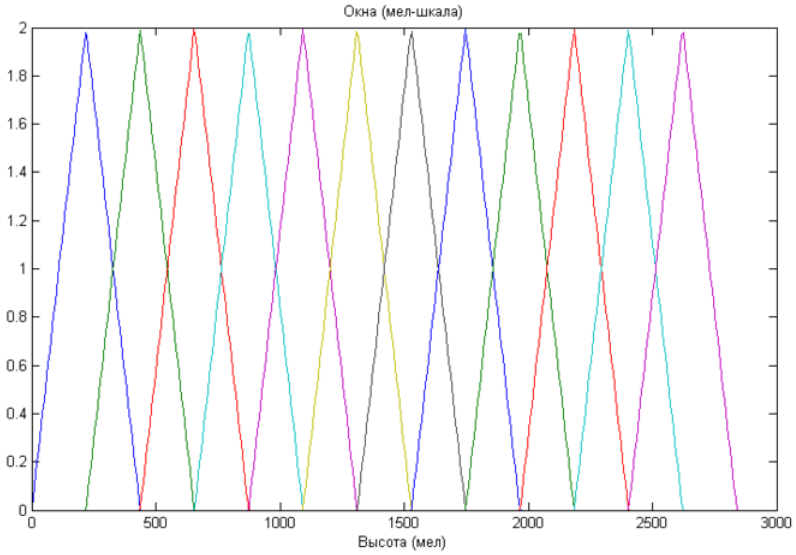

Рисунок 1.3 – Треугольные окна в мел-шкале

ходят применение в приложениях поиска музыкальной информации, таких как классификация жанров[11], меры сходства звука[12] и т. д.

Для получения коэффициентов используют следующий алгоритм:

1. Используя преобразование Фурье получить спектр исходного сигнала.
2. Располагаем треугольные окна равномерно на мел-шкале (см. рисунок 1.3).
3. Переводим треугольные окна в частотную шкалу по формуле:

$$f = 700(e^{\frac{m}{1127}} - 1) \tag{1.14}$$

4. Делаем свёртку спектра с каждым окном (см. рисунок 1.4) и берём логарифм спектра можности.
5. Полученные значения переводим во временную область дискретным косинусным преобразованием.
6. MFCC представляют собой амплитуды результирующего спектра.

1.5 Обзор аналогов

1.5.1 HOLO

Данный проект - это приложение, которое по анализу музыкальной коллекции позволяет составлять плейлисты похожих на заданные образцы композиции, а также визуализировать полученные данные о музыкальной коллекции[13]. Метод получение информационных признаков основан на преобразовании Фурье с последующим получением евклидового расстояния до набора готовых спектров, с последующим построением матрицы

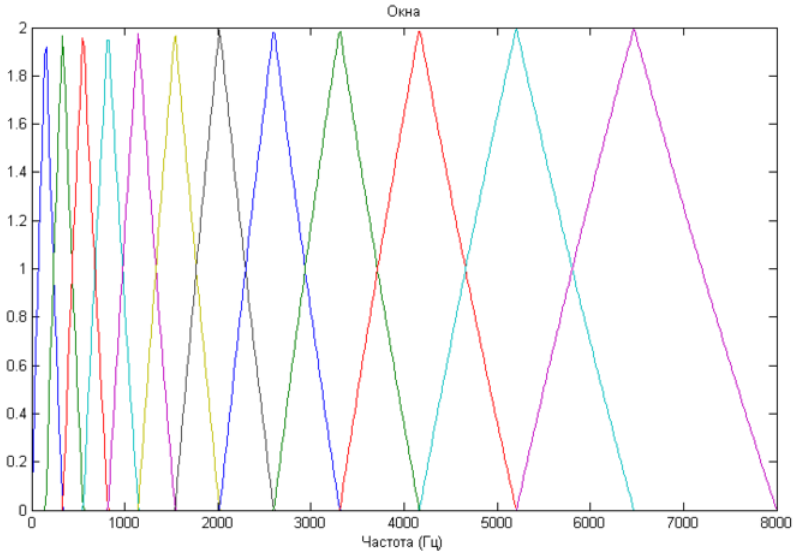

Рисунок 1.4 – треугольные окна в частотной шкале

переходов. Спектры носят служебную функцию опорных значений, и их форма выбрана по принципу отличаться друг от друга как можно сильнее, как по громкости, так и по корреляции графика частот. Формирование базы данных реализовано следующим образом:

1. Из файла извлекается фрагмент параметризуемой длины, начиная с 20% общей длины.
2. Фрагмент нарезается на параметризуемое количество окон с параметризуемым процентом перекрытия.
3. Каждое окно подвергается преобразованию Фурье, сглаживанию и очистке.
4. Имея некоторый набор заранее подготовленных центроидов, оценивается расстояние каждого окна до каждого из этих центроидов.
5. На основе расстояния окно маркируется порядковым номером ближайшей центроиды.
6. Строится матрица переходов между номерами центроид(рисунок 1.5).
7. Матрица переходов используется как вектор информационных признаков на основе которого происходит рекомендация.

Также в приложении реализовано визуализация результатов сканирования локальной музыкальной библиотеки.

Преимущества данного приложение:

- быстродействие;
- использует акустический анализ.

Минусы:

- нет взаимодействия с другими сервисами воспроизведения музыки;
- используется только 20% музыкального трека;

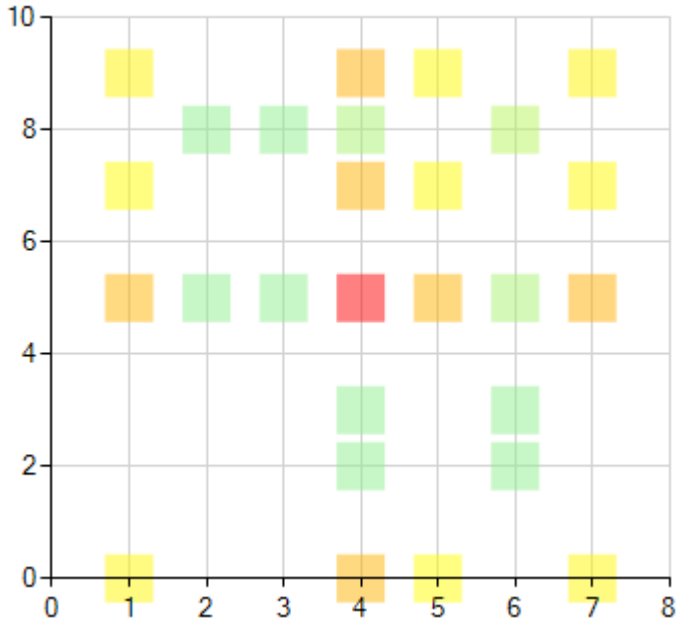

Рисунок 1.5 – Матрица переходов

- анализируются только MP3-файлы 44,1кГц 16 бит;

1.5.2 Athena-NeuroPlay

Данный проект - это приложение, которое использует генетический алгоритм, который на основе оценки пользователя качества исходной выборки строит топологию нейронной сети прямого распространения с несколькими скрытыми слоями [14].

Первый этап - нормализация данных. Для инструментов выделяются необходимые частоты . Также принимается в расчёт и чувствительность слуха в зависимости от частоты. Таким образом задача нормализации сводится к выделению некоторой информации о частотах, которая показывает:

- как часто в композиции звучит звук из данного диапазона частот;
- как громко он звучал;
- как долго он звучал;
- для каждого определенного диапазона частот (нужно разбить весь «слышимый» спектр на определенное число диапазонов(см. рисунок 1.6)).

Для выделения частотной насыщенности в треке на каждом временном интервале используется FFT. Временной интервал имеет размер 1024 сэмплов. На основе спектра получают следующие признаки: насыщенность звука определенными спектрами, частота возникновения различных спектров звука, его громкость, его длительность. Затем данные нормализуются.

На втором этапе используется нейронная сеть прямого распространения с 1024 входными нейронами и 24*7 выходными, где каждый выходной нейрон показывает «качество» трека для того или иного времени суток. В качестве исходной выборки используется плейлист, который делится на две

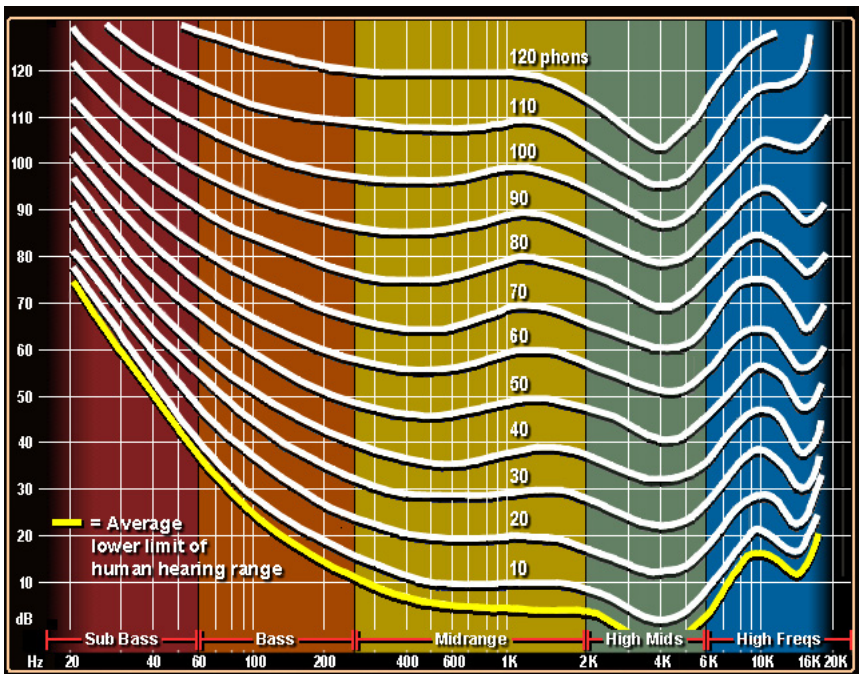

Рисунок 1.6 – Диапазоны частот

части: обучающая и контрольная. Размер обучающей и контрольной выборке устанавливается пользователем. При прослушивании трека из обучающей выборке пользователь ставит оценку от 0 до 1, где 0 - «плохой трек», а 1 - «хороший». Оценка влияет на все оценки по времени суток, но больше на ту, в которое время было поставлена оценка. После того как обучающая выборка промаркирована, идёт процесс обучения и настройки топологии нейронной сети. Топология нейронной сети определяется генетическим алгоритмом. Для этого берётся стартовая конфигурация в 10 слоев, в каждом по 100 нейронов, которая обучается за 1000 эпох, после определятся качество ее обучения. Качество обучения - это суммарная оценка контролирующей и обобщающей способности нейронной сети. На следующем этапе создаётся 10 конфигураций, каждая из которых является «мутантом» исходной, то есть у нее изменена в случайную сторону либо количество слоев, либо количество нейронов на каждом или некоторых слоях. Далее идёт обучение каждой конфигурации тем же способом, что и исходную. Выбирается лучшая из них по обобщающей способности и определяется как исходная. Данный процесс продолжаем до тех пор, пока не наступает такой момент, что мы не можем найти конфигурацию, которая обучается лучше чем исходная. Данную конфигурацию считаем лучшей, она оказалась способной лучше всех «запомнить» исходные данные и лучше всех предсказывает, то есть результат ее обучения наиболее качественный из возможных.

Из плюсов данного приложения стоит отметить:

- использует акустический анализ;
- запускается локально на компьютере пользователя.

Из недостатков:

- время работы;
- качество рекомендации.

1.5.3 Pandora Radio

Pandora (Пандора) - служба потокового воспроизведения музыки в Интернете, основанное на системе «Music Genome Project» [15]. Пользователь медиапроигрывателя Pandora выбирает музыкального исполнителя, после чего система ищет похожие композиции, используя около 400 музыкальных характеристик такие как жанр, тип инструментов, тип вокала, темп, синкопа, тональность, гармония и т. д. Используя функции «нравится» или «не нравится», слушатель может настроить «радиостанцию» по своему вкусу. В базе данных системы более миллиона композиций и более ста тысяч исполнителей[16]. Зарегистрированный пользователь может создать в своём профиле до 100 различных «радиостанций», транслирующих музыку в тех или иных жанрах. Медиапроигрыватель Pandora доступен пользователям с персональными компьютерами, смартфонами, планшетами с различными операционными системами.

Проект «Music Genome Project» - это набор более 450 атрибутов для описания песен и сложный математический алгоритм для их организации. Проект в настоящее время состоит из 5 суб-геномов : Pop / Rock, Hip-Hop / Electronica, Jazz, World Music и Classical. Песня представляется вектором, содержащим значения приблизительно для 450 «генов». Каждый ген соответствует характеристике музыки, например, пол ведущего вокалиста, уровень искажения на электрогитаре, тип фонового вокала и так далее. Рок и поп-песни имеют 150 генов, рэп-песни имеют 350 генов, а джазовые песни - приблизительно 400. Другие жанры музыки, такие как мировая и классическая музыка, имеют 300-450 генов. Система зависит от достаточного количества генов для получения полезных результатов. Учитывая вектор одной или нескольких песен, список других подобных песен построен с использованием того, что компания называет своим «алгоритмом сопоставления» [17]. Атрибуты для каждой песни выставляются музыкантом в процессе, который занимает от 20 до 30 минут на песню[18]. Десять процентов песен анализируются более чем одним музыкантом, чтобы обеспечить соответствие внутренним стандартам и статистическую надежность.

Преимущества данного сервиса:

- большая база данных музыки;
- имеются приложения для десктопов и мобильных платформ;
- быстродействие;
- взаимодействия с другими сервисами.

Из недостатков стоит отметить:

- сервис недоступен за пределами США, Австралии и Новой Зеландии;

- требуется регистрация;
- акустический анализ представлен не в чистом виде.

1.6 Выбор информационных образов для жанровой классификации музыкальных произведений

На основе анализа литературы были выбраны четыре типа информационных образов, которые будут извлекаться из музыкального трека:

- временной образ;
- спетральный образ;
- ритмический образ;
- мел-кепстральный образ.

Временные образ (здесь и далее представления музыкального трека как набора отсчётов будет называться сигналом).

1. Энергия сигнала, как мера яркости и громкости мелодии (см. формулу 1.3).

2. Количество переходов сигнала, через ноль, как мера зашумлённости (см. формулу 1.1).

3. Автокорреляция сигнала, как мера изменения резкости тембра (см. формулу 1.2.

Спектральный образ.

1. Среднее арифметическое взвешенное спектра. С точки зрения восприятия, оно имеет робастную связь с впечатлением «яркости» звука (см. формулу 1.5).

2. Линейная регрессия спектра, как мера отношения высокочастотной составляющей к низкочастотной составляющей звука и, следовательно, тембра (см. формулу 1.4).

3. Гладкость спектра, как мера гармоничности сигнала (см. формулу 1.6).

4. Дисперсия спектра относительно среднего взвешенного. С точки зрения восприятия, определяет «ширину» тембра (см. формулу 1.7).

5. Коэффициент асимметрии, как мера того, насколько искажён спектр относительно среднего взвешенного спектра, и, следовательно наклон к высоким или низким частотам (см. формулу 1.8).

6. Энтропия Винера или мера спектральной плоскостности. Определяет зашумлённость сигнала. Чем меньше значение, тем больше спектральной мощности сосредоточено в относительно небольшом числе полос (см. формулу 1.12).

7. Энергетическое спектральное окно по уровню 0,85, как мера спектральной формы (см. формулу 1.9)

8. Коэффициент амплитуды (см. формулу 3.1).

Ритмический образ. Признаки считаются по коррелограмме.

1. Амплитуда первого пика.

2. Отношение частот первого пика к частоте второго пика.
3. Амплитуда второго пика.
4. Отношение частот второго пика к частоте третьего пика.
5. Амплитуда третьего пика.
6. Отношение частот третьего пика к частоте четвёртого пика.
7. Амплитуда четвёртого пика.
8. Частота первого пика.

Мел-кепстральный образ представляет из себя 16 мел-кепстральных коэффициентов.

2 СИСТЕМНОЕ ПРОЕКТИРОВАНИЕ

Изучив теоретические аспекты разрабатываемого модуля и выработав список требований необходимых для разработки модуля, разбиваем систему на компоненты. Компоненты в виде блоков и их взаимосвязи указаны на чертеже. В разрабатываемом модуле можно выделить следующие блоки:

- модуль чтения музыкального произведения;
- модуль препроцессинга и нарезки музыкального трека на фрагменты;
- модуль получения частотно-временного представления сигнала;
- модуль извлечения информационных образов;
- модуль обработки информационных образов;
- база данных информационных образов;
- модуль жанровой классификации музыкального произведения.
- модуль визуализация.

Структурная схема, иллюстрирующая перечисленные блоки и связи между ними приведена на чертеже ГУИР.400201.004 С1. Для решения задачи выделения спектральных, временных и иных признаков из музыкального трека будут использоваться с ледующие блоки:

- модуль препроцессинга и нарезки музыкального трека на фрагменты;
- модуль получения частотно-временного представления сигнала;
- модуль извлечения информационных образов;
- модуль обработки информационных образов.

Для решения задачи проверки значимости признаков путём использовании их в задаче жанровой классификации будет использоваться модуль жанровой классификации музыкального произведения.

Для решения задачи визуализации данных алгоритмом t-SNE используется модуль визуализации.

Модуль чтения музыкального произведения состоит из двух частей. Первая часть это консольная утилита Lame. Lame свободное приложение для кодирования аудио в формат MP3 (MPEG-1 audio layer 3) и декодирования аудио в WAV формат, который наиболее удобен для чтения и представления музыкальных треков в виде массива. Библиотека Scipy позволяет преобразовать WAV аудио в Numpy массив.

Модуль препроцессинга обеспечивает первоначальную обработку музыкальных треков. На вход модуля музыкальный трек подаётся как набор сэмплов (NumPy массив). В данном блоке происходит нормализация, фильтрация и, если необходимо, приведения стереозвука к монозвуку .

Модуль получения частотно-временного представления сигнала представляет собой набор преобразований такие как оконное преобразование Фурье и вейвлеты, которые представляются в виде многомерного NumPy - массива.Так подобные вычисления требуют высокой производительности,

то для более эффективного вычисления используются параллельные вычисления на центральном процессоре.

Модуль извлечения информационных образов. Данный модуль представляет из себя набор методов получения информационных образов из временной и частотной области музыкального произведения. Также в этом модуле считаются мел-кепстральные коэффициенты. На вход данному модулю подаётся многомерные Numpy массивы. На выходе получается одномерный Numpy массив информационных признаков для каждого музыкального трека.

Модуль обработки информационных образов. В данном модуле признаки нормализуются по МО и СКО. Удаляются выбросы и некорректные значения.

аза данных информационных образов хранит в себе результат работы всего модуля и хранит в себе набор информационных образов музыкальных треков и принадлежность к тому или иному кластеру.

При выборе базы данных были сформулированы следующие требования:

- производительность;
- объектный язык запросов;
- возможность параллельной записи и чтения.

Так как главный модуль является частью сервиса рекомендации музыки, то появляется дополнительные требования к базе данных:

- маштабируемость;
- репликация;
- балансировка нагрузки.

С учётом всех этих требований и того факта, что данные для хранения представляют собой простую структуру, выбор пал на базу данных MongoDB. MongoDB — документоориентированная система управления базами данных (СУБД) с открытым исходным кодом, не требующая описания схемы таблиц. Классифицирована как NoSQL, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Имеется подробная и качественная документация, большое число примеров и драйверов под популярные языки Java, JavaScript, Node.js, C++, C#, PHP, Python, Perl, Ruby. MongoDB может работать с набором реплик. Набор реплик состоит из двух и более копий данных. Каждый экземпляр набора реплик может в любой момент выступать в роли основной или вспомогательной реплики. Все операции записи и чтения по умолчанию осуществляются с основной репликой. Вспомогательные реплики поддерживают в актуальном состоянии копии данных. В случае, когда основная реплика дает сбой, набор реплик проводит выбор, который из реплик должен стать основным. Второстепенные реплики могут дополнительно является источником для операций чтения. MongoDB масштабируется горизонтально используя шардинг. Пользователь

выбирает ключ шарда, который определяет как данные в коллекции будут распределены. Данные разделятюся на диапазоны (в зависимости от ключа шарда) и распределятся по шардам. Из преимуществ MongoDB:

1. Объектный язык запросов.
2. Поддержка индексации.
3. Поддержка Map/Reduce для распределенных операций над данным.
4. Документы, не требующие определения схемы. Одно из самых важных преимуществ. Преимущество заключается в том, что нет нужды хранить пустые ячейки данных в каждом документе.
5. Поддержка сложных массивов. Каждый элемент массива может представлять из себя объект.
6. Поддержка шардинга на уровне платформы.
7. Атомарность гарантируется только на уровне целого документа, то есть частичного обновления документа произойти не может.
8. Любые данные, которые считываются одним клиентом, могут параллельно изменяться другим клиентом.

СУБД управляет наборами JSON-подобных документов, хранимых в двоичном виде в формате BSON. Хранение и поиск файлов в MongoDB происходит благодаря вызовам протокола GridFS.

Модуль жанровой классификации необходим для проверки значимости выделенных образов. Для этого используется набор стандартных алгоритмов классификации из библиотеки Scikit-learn с параметрами по умолчанию. На вход модуля поступает двумерный Numpy-массив. На выходе матрица ошибок. В качестве исходной выборки используется репозиторий музыки GZTAN.

Модуль визуализации визуализирует данные с помощью алгоритма t-SNE. На вход подаётся двумерны Numpy-массив. На выходе изображение в формате JPEG.

Для реализации модулей был выбран язык программирования Python, так как для него существует множество библиотек выполняющих математические расчеты, и облегчающих решение задач связанных с анализом данных и машинным обучением. В работе используются библиотека NumPy - это расширение языка Python, добавляющее поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых математических функций для операций с этими массивами. Также существует библиотека SciKit-learn, которая содержит реализацию алгоритмов машинного обучения инструменты для работы с данными. В дипломной работе также используется библиотека Skipy, которая предназначенная для выполнения научных и инженерных расчётов.

3 ФУНКЦИОНАЛЬНОЕ ПРОЕКТИРОВАНИЕ

3.1 Алгоритм работы модуля выделения информационных образов из музыкального произведения

На ход модуля чтения музыкального произведения подаётся путь к музыкальному произведению. Файл с расширением ".wav" считывается в оперативную память в формате массива отсчётов. Файл с иным расширением конвертируются в WAV формат.

В модуле препроцессинга и нарезки от музыкального трека отрезается 10% длины с начала и конца трека. В случае, если трек является многоканальным, то он приводится к одноканальному посредством чередованием правого и левого канала. Данные нормализуются по МО и СКО. Затем используется экспоненциальное сглаживание с коэффициентом сглаживания 0,99. Результат нарезается на фрагменты по 5 секунд с перекрытием в 0,5 секунды.

В модуле получения частотно-временного представления используется оконное преобразование Фурье с окном Хэмминга шириной в 10 миллисекунд. Также в модуле для получения ритмического образа используется вейвлет Добеши каскадным алгоритмом с количеством каскадов 4. Результат каждого каскада сглаживается экспоненциальным сглаживанием с коэффициентом сглаживания 0,97, передискретизируется с уменьшением дискретизации в 16 раз, нормализуется по МО. Затем результаты поэлементно складываются и вычисляется автокорелляция. Результатом работы модуля является спектрограмма и коррелограмма.

В модуле выделения информационных образов из временного представления сигнала выделяется временной, из спектрограммы - спектральный образ по каждому срезу спектрограммы, из коррелограммы - ритмический. Мел-кепстральные коэффициенты считаются по окнам в 5 миллисекунд.

В модуле обработки информационных образов по всем образам кроме временного считается МО, СКО, коэффициент ассиметрии, коэффициент эксцесса.

В базу данных записывается информационный образ в формате: название произведения, номер фрагмента и результат работы модуля обработки информационных образов.

В модуле классификации информационных образов используются набор стандартных алгоритмов классификации из библиотеки scikit-learn с параметрами по умолчанию:

а) метод ближайших соседей с k = 3;
б) метод опорных векторов с полиномиальным ядром;
в) дерево принятия решений;
г) случайный лес;

д) нейронная сеть прямого распространения с 1000 нейронов на скрытом слое;

е) adaBoost;

ж) наивный баесовский классификатор;

з) квадратичный дискриминант.

Для оценки качества классификации использовался скользящий контроль количеством разбиений равным 10 и алгоритмом разбиения stratified k-fold. Также для каждого классификатора строится матрица ошибок.

В модуле визуализации используется алгоритм t-sne.Это нелинейный метод уменьшения размерности, который особенно хорошо подходит для вложения высокоразмерных данных в пространство двух или трех измерений, которое затем можно визуализировать на диаграмме рассеяния. В частности, он моделирует каждый высокоразмерный объект с помощью двух- или трехмерной точки таким образом, что аналогичные объекты моделируются соседними точками, а разнородные объекты моделируются удаленными точками. Данные визуализируются в 2д и 3д графики.

3.2 Реализация внутренних модулей

Рассмотрим в деталях функциональные части системы: для этого произведем детальный анализ компонентов, модулей, составляющих их классов и отдельных методов, реализующих логику программы.

Структурно система подразделяется на ряд модулей , объединенных под общим модулем с названием `MainModule`. Дальнейшее разделение ведется по следующим модулям:

1. Модуль чтения музыкального произведения - `WavModule`.
2. Модуль препроцессинга и нарезки - `PreprocessingModule`.
3. Модуль получения частотно-временного представления сигнала - `SpectralTransformerModule`.
4. Модуль извлечения информационных образов - `FeatureExtractorModule`.
5. Модуль обработки информационных образов - `FeatureProcessingModule`.
6. База данных информационных образов - `DatabaseModule`.
7. Модуль жанровой классификации музыкального произведения - `GenreClassificationModule`.
8. Модуль визуализации - `VisualizeDataModule`.

3.2.1 Класс `Track`

Класс `Track` является абстрацией для хранения трека в виде массива отсчётов, частоты дискретизации и метаинформации о треке.

Класс `Track` имеет следующие поля:

- Поле `data` – массив ndarray, который хранит в трек в виде отсчётов.
- Поле `sample_rate` – частота дискретизации трека.
- Поле `label` – поле хранящее метаинформацию о треке.

3.2.2 Класс `SpectralTrack`

Класс `SpectralTrack` наследуется от класса `Track` и является абстрацией для хранения спектра трека и его коррелограммы

Класс `Track` имеет следующие поля:

- Поле `spectra_data` – массив ndarray, который хранит спектрограмму трека, полученную с помощью оконного преобразования Фурье.
- Поле `percussion_data` – масств ndarray, который хранит коррелограмму трека.

3.2.3 Класс `TrackModel`

Класс `TrackModel` является абстракцией для хранения временного, спектрального, ритмического и мел-кепстрального образа.

Класс `TrackModel` имеет следующие методы и поля:

- Поле `timing_features` – массив ndarray, который хранит временной образ трека.
- Поле `spectral_features` – масств ndarray, который хранит спектральный образ трека.
- Поле `percussion_features` – массив ndarray, который хранит ритмический образ трека.
- Поле `mfcc_features` – массив ndarray, который хранит в себе мел-кепстральный образ трека.
- Поле `label` – поле хранящее метаинформацию о треке .
- `to_vector()` – метод, который конкатенирует образы. И возвращает одномерный массив типа ndarray.

3.2.4 Класс `WavModule`

Класс `WavModule` предназначен для преобразования MP3 к формату WAV и считывании данных в Numpy-массив.

Класс `WavModule` имеет следующие методы:

- `create_wav(file_name)` процедура, которая принимает на вход путь к MP3-файлу и с помошью стандартной библиотеки Python вызывает либо bash-скрипт в случае запуска в операционной системе Linux, либо batch-скрипт в случае запуска в операционной системе Windows, который использует консольное приложение Lame для преобразования MP3-файла в WAV формат
- `read_wav(filename, label)` - метод, который считывает файл формата WAV и сохраняет данные в класс `Track`

3.2.5 Класс `PreprocessingModule`

Класс `PreprocessingModule` предназначен для первичной обработки трека, которая включает в себя приведения стерео звука к моно, нормализация по МО и СКО, удаления заданного процента трека с начала и с конца, экспоненциальное сглаживание и нарезка трека на фрагменты. В конструкторе задаётся коэффициент сглаживания, процент пересечения фрагментов, процент отсечения трека с начала, процент отсечения трека с конца, размер фрагмента в секундах.

Класс `PreprocessingModule` имеет следующие методы и поля:

- Поле `alpha` – коэффициент сглаживания, который может принимать значения от 0 до 1.
- Поле `overlap` – процент пересечения фрагментов.
- Поле `cut_start` – процент отсечения трека с начала трека
- Поле `cut_end` – процент отсечения трека с конца трека
- Поле `frame_size_sec` – размер фрагментов в секундах
- `stereo_to_mono(track)` – метод, который преобразует стерео звук к моно с помощью чередования правого и левого канала
- `scale(track)` – метод, который нормализует данные по МО и СКО
- `filter(track)` – метод, который преобразовывает данные искользуя экспоненциальное сглаживание:

$$s_t = \begin{cases} c_1, & t_1 = 0. \\ s_t + \alpha * (c_t - s_{t-1}), & t > 0 \end{cases} \tag{3.1}$$

где - α - коэффициент сглаживание заданные в поле `alpha`

- `cutting(track)` – метод, который отсекает процент данных трека с начала и конца, который задаётся полями `cut_start` и `cut_end`
- `framing(track)` – метод, который нарезает трека на фрагменты длинной заданной полем `frame_size_sec` и перекрытием заданным полем `overlap`.

3.2.6 Класс `SpectralTransformer`

Класс `SpectralTransformer` предназначен для получения спектрально-временного представления трека, а также для получения коррелограммы. Спектрально-временное представление получается путём оконного преобразования Фурье (см. формулу 3.2) с окном Хемминга размером 10 миллисекунд (см. формулу 3.3)

$$F(m, \omega) = \sum_{n=-\infty}^{\infty} f[n] w[n - m] e^{-j\omega n} \tag{3.2}$$

$$w_i = 0.54 - 0.46 \cos \frac{2\pi i}{n-1} \tag{3.3}$$

Также в модуле получает ритмические(перкуссионные) признаки.

Класс `SpectralTransformer` имеет следующие методы и поля:

• Поле `alpha` – коэффициент сглаживания, который может принимать значения от 0 до 1.

• Поле `window` – массив, который содержит в себе окно Хемминга.

• Поле `level` – количество выходов из каскадов в дискретном вейвлет преобразовании.

• Поле `rate` – частота передискретизации.

• `short_time_fourier(track)` – метод, который преобразует трек оконным преобразованием Фурье с окном Хемминга, возвращает двумерный массив амплитуд типа `ndarray`.

• `wavelet_daubechies(data)` – метод, который делает дискретное вейвлет преобразования Добеши.

• `filter(track)` – метод, который преобразовывает данные искользуя экспоненциальное сглаживание.

• `resampling(data)` – метод, который передискретизирует данные в `rate` раз

• `normalize_and_sum(track)` – метод, который нормализует по МО и суммирует выходные коэффициенты каскадов.

3.2.7 Класс `FeatureExtractror`

Класс `FeatureExtractror` предназначен для выделения признаков из временной, спектральной и ритмической области.

Класс `FeatureExtractror` имеет следующие методы и поля:

• Поле `time_feature_models` – массив методов, которые выделяют признаки из временной области трека. Методы представляют из себя наследников классов `TimingFeature`.

• Поле `spectre_feature_models` – массив методов, которые выделяют признаки из спектральной области трека. Методы представляют из себя наследников классов `SpectralFeature`.

• Поле `results` – словарь с ключём – тип метода выделения, и значением – результат метода выделения. Типы ключа являются наследниками класса `FeatureExtractrorModel`.

• Поле `nceps` – количество мел-кепстральных коэффициентов.

• `extract_feature(track)` – метод, который извлекает признаки из временной области, используя методы сохранённые в `time_feature_models`.

• `extract_percussion_feature(track)` – метод, который извлекает ритмические признаки.

• `extract_mfcc(track)` – метод, который извлекает мел-кепст-

ральные коэффициенты из каждого среза спектрограммы.

• `extract_spectra_feature(track)` – метод, который извлекает спектральные признаки из каждого среза спектрограммы, используя методы сохранённые в `spectral_feature_models`.

3.2.8 Класс `FeatureExtractorModel`

Класс `FeatureExtractorModel` абстрактный класс, который используется в качестве базового для классов, которые извлекают признак из временной или из спектральной области.

Класс `FeatureExtractorModel` имеет следующие методы:

• `get(data, params)` – метод, который принимает на вход ndarray и массив параметров, а на выход выдаёт признак типа float. В данном классе метод абстрактный

• `normalize(result, data))` – метод, который нормализует данные про размер входного массива

3.2.9 Класс `SpectralFeature`

Класс `SpectralFeature` абстрактный класс, который используется в качестве базового для классов, который извлекают признак из спектральной области. Класс наследуется от `FeatureExtractorModel`.

3.2.10 Класс `TimingFeature`

Класс `TimingFeature` абстрактный класс, который используется в качестве базового для классов, который извлекают признак из временной области. Класс наследуется от `FeatureExtractorModel`.

3.2.11 Класс `Energy`

Класс `Energy` класс, который получает значение энергии сигнала из временной области (см. формулу 1.3). Класс наследуется от `TimingFeature`.

Класс `Energy` имеет следующие метод `get(data, params)` – метод, который принимает на вход массив отсчётов, а на выход выдаёт энергию сигнала.

3.2.12 Класс `ZeroCrossingRate`

Класс `ZeroCrossingRate` класс, который получает значение количества переходов сигнала через ноль из временной области (см. формулу 1.1). Класс наследуется от `TimingFeature`.

Класс ZeroCrossingRate имеет метод itemget(data, params) – метод, который принимает на вход массив отсчётов, а на выход выдаёт количества переходов сигнала через ноль .

3.2.13 Класс Autocorrelation

Класс Autocorrelation класс, который получает значение автокорреляцию первого рода из временной области (см. формулу 1.2). Класс наследуется от TimingFeature.

Класс Autocorrelation имеет метод get(data, params) – метод, который принимает на вход массив отсчётов, а на выход выдаёт автокорреляцию первого рода.

3.2.14 Класс SpectralCentroid

Класс SpectralCentroid класс, который получает значение арифметического среднего взешенного из спектра (см. формулу 1.5). Класс наследуется от SpectralFeature.

Класс SpectralCentroid имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход значение арифметического среднего взвешенного.

3.2.15 Класс SpectralSmoothness

Класс SpectralSmoothness класс, который получает значение гладкости спектра (см. формулу 1.6). Класс наследуется от SpectralFeature.

Класс SpectralSmoothness имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход значение гладкости спектра.

3.2.16 Класс LinearRegression

Класс LinearRegression класс, который получает значение линейной регрессии спектра (см. формулу 1.4). Класс наследуется от Spect-ralFeature.

Класс LinearRegression имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход значение линейной регрессии спектра.

3.2.17 Класс SpectralSpread

Класс SpectralSpread класс, который получает значение дисперсии спектра относительно арифметического среднего взвешенного (см. формулу 1.7). Класс наследуется от SpectralFeature.

Класс SpectralSpread имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр и params, где первым элементом идёт вычисленное арифметическое среднее взвешенное , а на выход выдаёт значение дисперсии спектра относительно арифметического среднего взвешенного.

3.2.18 Класс SpectralDissymmetry

Класс SpectralDissymmetry класс, который получает коэффициент асимметрии спектра (см. формулу 1.8). Класс наследуется от SpectralFeature.

Класс SpectralDissymmetry имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр и массив params, где первым элементом идёт вычисленное арифметическое среднее взвешенное , а на выход выдаёт значение коэффициент асимметрии спектра.

3.2.19 Класс Rolloff

Класс Rolloff класс, который получает энергетическое спектральное окно по уровню 0,85 (см. формулу 1.9). Класс наследуется от SpectralFeature.

Класс Rolloff имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход энергетическое спектральное окно по уровню 0,85.

3.2.20 Класс SCF

Класс SCF класс, который получает коэффициент амплитуды (см. формулу 1.12) показывающий отношение пиковых значений к эффективному значению. Класс наследуется от SpectralFeature.

Класс SCF имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход отношение пиковых спектра значений к эффективному значению.

3.2.21 Класс SFM

Класс SFM класс, который энтропию Винера или коэффициент спектральной плоскостности (см. формулу 3.1). Класс наследуется от SpectralFeature.

Класс SFM имеет метод get(data, params) – метод, который принимает на вход массив содержащий спектр, а на выход коэффициент спектральной плоскостности.

3.2.22 Класс FeatureProcessing

Класс FeatureProcessing предназначен для вычисления статистических признаков. Так как спектральные признаки и мел-спектральные коэффициенты вычисляются по каждому срезу спектрограммы, то для сокращения размера информационного вектора вычисляются МО, СКО, коэффициент асимметрии и коэффициент эксесса. Для каждого спектрального признака или мел-спектрального коэффициента эти параметры вычисляются независимо.

Класс FeatureProcessing имеет следующие методы и поля:

- Поле with_mean – флаг использования МО.
- Поле with_std – флаг использования СКО.
- Поле with_skew – флаг использования коэффициента ассиметрии.
- Поле with_kurtosis – флаг использования коэффициента эксесса.
- mean(data) – метод, который возвращает МО для каждого спектрального признака и мел-спектрального коэффициента.
- std(data) – метод, который возвращает СКО для каждого спектрального признака и мел-спектрального коэффициента.
- skew(data) – метод, который возвращает коэффициент ассиметрии для каждого спектрального признака и мел-спектрального коэффициента.
- kurtosis(data) – метод, который возвращает коэффициент эксесса для каждого спектрального признака и мел-спектрального коэффициента.
- process_feature(track) – метод, который возвращает класс TrackModel с теми вычилеными признаками, которые были установлены флагами в конструкторе класса.

3.2.23 Класс GenreClassificationModule

Класс GenreClassificationModule предназначен для жанровой классификации и получения матрицы ошибок, а также оценки методов классификации методом перекрёстной проверки. Также есть возможность отобразить и сохранить матрицу ошибок.

Класс GenreClassificationModule имеет следующие методы и поля:

- Поле classifiers – ассоциативный массив, где ключ – названия метода классификации, а значения – массив методов классификации, который состоит из классов наследуемых от абстрактного класса библиотеки sklearn ClassifierMixin. и поддерживать следующий интерфейс:

1. fit(X, y) – метод для обучения классификатора, где X – набор образов, а y – их классы.

2. `predict(X)` – метод, который возвращает класс образа.

3. `score(X, y)` – метод, котрый возвращает среднюю точность по данным тестовых данных и их классов.

• Поле `labels_name` – массив строк, который хранит в себе названия классов.

• Поле `cv` – количество разбивок исходной выборки, на обучающую подвыборку и контрольную подвыборку.

• `cross_val_score(clf, data, labels, cv)` – статический метод, который принимает на вход метод классификации, выборку образов, их классы и количество разбивок. Возвращает оценку методом перекрёстной проверки.

• `cross_validation_predict(clf, data, labels, cv)` – статический метод, который имеет подобную сигнатуру как и `cross_val_score`, но возвращает для каждого элемента на входе предсказание, которое было получено для этого элемента, когда оно находилось в тестовом наборе.

• `plot_confusion_matrix(cnf_matrix, clf_name, show)` – метод, который принимает матрицу ошибок, название классов и флаг отображения матрицы в новом окне. Метод созраняет матрицу ошибок в файл в формате pdf.

• `classify(data, labels, meta)` – метод, который принимает на вход выборку образов, их классы и мета информациию о треке, которая не участвует в классификации. А на выходе ассоциативный массив где ключ – это название метода классификации, а значение – массив, где хранится МО оценки классификации перекрёстной проверкой, СКО оценки классификации перекрёстной проверкой и матрица ошибок. Вычисления каждого метода классификации происходит в отдельном процессе.

3.2.24 Класс `VisualizeDataModule`

Класс `VisualizeDataModule` предназначен для визуализации данных в двухмерной и трёхмерной плоскости. Для уменьшения пространства признаков используется алгоритм t-sne. Результаты визуализации сохраняются в pdf-файл.

Класс `VisualizeDataModule` имеет следующие методы и поля:

• `dimension_reduction(data, n_component, reduction_method)` – метод, который уменьшает размерность выборки. На вход принимает выборку, новую размерность и метод уменьшения размерности(либо t-sne, либо pca). на выход возвращает выборку с заданной размерности и нормализованной от нуля до единицы.

• `plot_2d(data, labels, genre_list, show, reduction_method)` – метод, который создаёт и сохраняет визуализацию выборки в двухмерном измерении. На вход принимает массив информационных обра-

зов, их классы, название класов, флаг отображения в отдельном окне, метод уменьшения размерности. Результат визуализации сохраняется в pdf-файл с название 2d.pdf.

• `plot_3d(data, labels, genre_list, show,reduction-_method)` – метод, который создаёт и сохраняет визуализацию выборки в трёхмерном измерении. На вход принимает массив информационных образов, их классы, название класов, флаг отображения в отдельном окне, метод уменьшения размерности. Результат визуализации сохраняется в pdf-файл с название 2d.pdf.

3.2.25 Класс `DatabaseModule`

Класс `DatabaseModule` предназначен для сохранения `TrackModel` в базу данных MongoDB. Для этого в конструкторе объекта нужно передать ip-адрес базы данных и порт на которой работает MongoDB.

Класс `DatabaseModule` имеет следующие методы и поля:

• `track_model_to_dict(track)` – метод, который преобразует класс `TrackModel` в ассоциативный контейнер , где ключ – название поля, а значение – значения поля класса. `itemstore(track)` – метод, который сохраняте класс `TrackModel` в базу данных в формате ассоциативного контейнера, который описан выше.

4 РАЗРАБОТКА ПРОГРАММНЫХ МОДУЛЕЙ

При разработке системы одними из наиважнейших требований к исходному коду являются его расширяемость и поддерживаемость. Реализация программных модулей с учетом этих требований приводит к простоте расширения функционала в критических местах, обеспечению разделенности и независимости компонентов системы, что улучшает их тестируемость и в целом позволяет добиться реализации более стабильной и простой в понимании кодовой базы.

4.1 Вычисления временных и спектральных признаков

В классе `FeatureExtractor` для упрощения добавления способов получения признаков, которые могут иметь зависимости между собой, используется абстрактный класс `FeatureExtractorModel` для декларации интерфейса и два абстрактных класса для различения признаков по временным и спектральным - `TimingFeature` и `SpectralFeature` соответственно. В сигнатуре метода `get(data, params)` `params` – это параметр, через который передаётся вычисленный признак и который необходим для вычисления текущего.

```
class FeatureExtractorModel:
   __metaclass__ = ABCMeta

   @abstractmethod
   def get(self, data, params=None):
      pass

   def check(self, data):
      if not isinstance(data, np.ndarray):
         raise TypeError("input is not array")

   def normalize(self, result, data):
      return result / float(len(data))
```

Так как вычисления одних признаков зависит от вычисления других, то задача получения всех признаков становится задачей выполнения графа. Поэтому для описание графа используется ассоциативный массив, где ключ – тип метода выделения признака, который наследуется либо от `TimingFeature`, либо от `SpectralFeature`, а значение массив таких же типов результаты которых передаётся типу ключа.

```
models = {
```

```
    Energy: [],
    ZeroCrossingRate: [],
    Autocorrelation: [],
    SpectralCentroid: [],
    SpectralSmoothness: [],
    SpectralSpread: [SpectralCentroid],
    SpectralDissymmetry: [SpectralCentroid],
    Rolloff: [],
    LinearRegression: [],
    SFM: [],
    SCF: []
}
        ...
```

Как видно из примера, классам `SpectralSpread` и `SpectralDissymmetry` требуется результат вычисления класса `SpectralCentroid`. Для получения всех признаков класс `FeatureExtractor` интерактивно вычисляет признаки, которые не зависят от результат вычисления других признаков, а потом те, которые зависят только от одного и так далее. Значение признаков заносится в ассоциативный массив, где ключ – это тип, а значение – это результат вычисление данного признака.

```
    def eval_models(self, extractors, data):
        max_length = np.amax(map(lambda x: len(x),
            extractors.values())) + 1)
        for i in range(max_length):
            for feature_extractor in filter(lambda x: len
                (extractors[x]) == i, extractors):
                self.results[feature_extractor] = \
                    feature_extractor() \
                        .get(data,
                             map(lambda x: self.results[x]
                                 if x in self.results else
                                 None,
                                 extractors[feature_extractor
                                 ]))
        ...
```

4.2 Параллельное распараллеливание вычислений в модуле `GenreClassificationModule`

В модуле классификации `GenreClassificationModule` для ускорения классификации было использованно распараллеливание на несколько

процессов. Для это использовалась библиотека joblib, которая позволяет выполнять циклы на всех ядрах процессора. Главным ограничением это библиотеки – распараллелить можно только функцию вне класс. Поэтому вся логика классификации была вынесена в отдельную функцию.

```
def classify(self, data, labels, meta):
    temp = Parallel(n_jobs=CPU_COUNT)(
       delayed(classify_p)
       (self.classifiers[name], name, data, labels,
         self.cv, meta) for name in self.classifiers
    )
    result = dict()
    for i in temp:
       result.update(i)
    return result
```

Результатом работы функции `classify_p` является ассоциативный массив, где ключ – название метода классификации, а значение представляет собой массив, который содержит в себе:

- МО оценки перекрёстной проверки;
- СКО оценки перекрёстной проверки;
- матрицу ошибок классификации.

Так как функция `Parallel` возвращает массив результатов функции, которую распараллелили, то необходимо собрать массив одноэлементных словарей в одни большой массив.

4.3 Отображение матрицы ошибок классификации

В классе `GenreClassificationModule` для отображения матрицы ошибок классификации матрицы используется библиотека Matplotlib, которая позволяет отображать многие виды графиков и диаграмм. Для большей наглядности цвет фона ячейки матрицы зависит от значения, которое находится внтури. В качестве цвета фона используется синий цвет в градации от белого до почти чёрного.

```
plt.imshow(cm, interpolation='nearest', cmap=cmap)
plt.colorbar()
```

На вход функции отображения подаётся матрица ошибок с количеством распознаных образом. Для нормализации данных для каждого ряда считается сумма и значение каждого элемента ряда делится на сумму, умножается на 100 и округляется.

```
cm = np.round(cm.astype('float') /
     cm.sum(axis=1)[:, np.newaxis] * 100)\
```

```
    .astype('int')
```

Определяется порог при котором цвет шрифта внутри клетки меняется с чёрного на белый.

```
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]),
  range(cm.shape[1])):
   plt.text(j, i, cm[i, j],
      horizontalalignment="center",
      color="white" if cm[i, j] > thresh else "
        black")
```

5 РЕЗУЛЬТАТЫ ЖАНРОВОЙ КЛАССИФИКАЦИИ

Для проверки значимости выделенных образов было принято решения проверить их на задаче жанровой классификации.

В качестве исходной выборки был использован репозиторий GZTAG. Набор данных состоит из 1000 звуковых дорожек каждые по 30 секунд. Он содержит 10 жанров, каждый из которых представлен 100 треками. Все треки - это все 220-мегагерцовые Mono 16-битные аудиофайлы в формате .wav.

Для решения задачи классификации была использована библиотека scikit-learn – бесплатная библиотека для машинного обучения с открытым исходным кодом для языка программирования Python. Все методы классификации использованы с параметрами по умолчанию.

5.1 AdaBoost с деревьями принятия решений

AdaBoost zвляется мета-алгоритмом, в процессе обучения строит композицию из базовых алгоритмов обучения для улучшения их эффективности. AdaBoost является алгоритмом адаптивного бустинга в том смысле, что каждый следующий классификатор строится по объектам, которые плохо классифицируются предыдущими классификаторами.

AdaBoost вызывает слабый классификатор в цикле. После каждого вызова обновляется распределение весов, которые отвечают важности каждого из объектов обучающего множества для классификации. На каждой итерации веса каждого неверно классифицированного объекта возрастают, таким образом новый классификатор «фокусирует своё внимание» на этих объектах.

В качестве слабого классификаторы было использованно дерево принятие решений. Максимальное количество слабых классификаторов установлено в количестве 50. В качестве алгоритма обучения использовался алгоритм SAMME.

Как показывает рисунок 5.1 данный классификатор плохо справляется с задачей классификации. Классифицировались фрагменты по 5 секунд с перекрытием в 0,5 секунд. Из жанров лучше всего классифицировались классическая музыка – 70 %, металл – 47 % и хипхоп – 49 %.

Оценка перекрёстной проверки с десятью разбиениями – 28 % со средней квадратичным отклонением 3,8 %.

Для задачи жанровой классификации данный метод с текущими параметрами и с теми выделенными образами для решения задачи жанровой классификации не подходит.

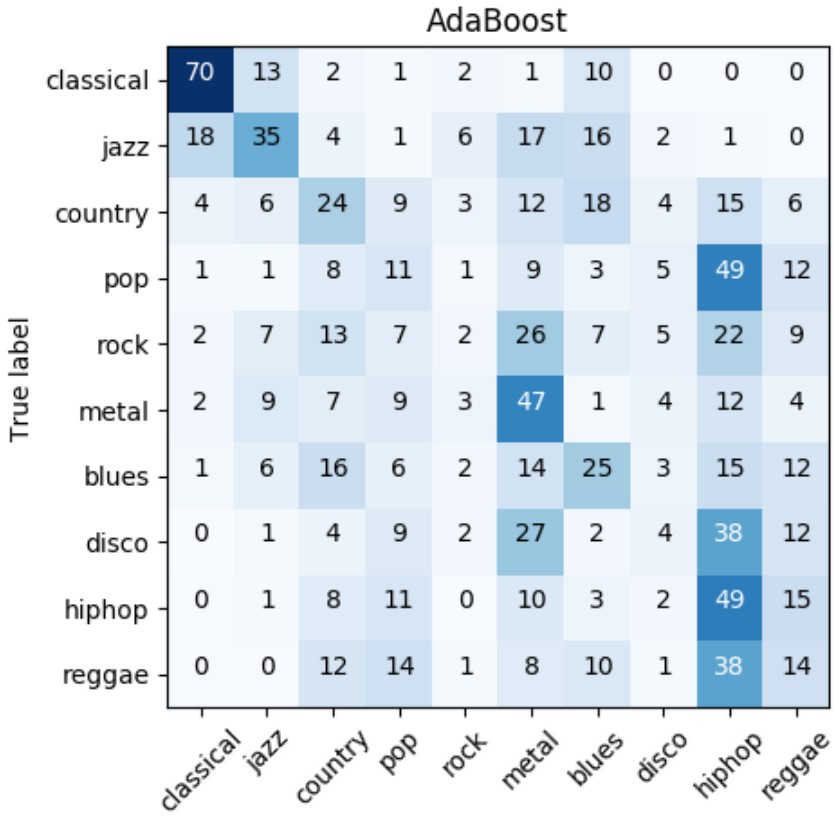

Рисунок 5.1 – Матрица ошибок для фрагмента трека полученная с помощь метода классификации AdaBoost SAMME, где используется «комитет» деревьев принятия решений.

5.2 Дерево принятия решений

Деревья принятия решений - это непараметрический контролируемый метод обучения, используемый для классификации и регрессии. Цель состоит в том, чтобы создать модель, которая предсказывает значение целевой переменной путем изучения простых правил принятия решений, выведенных из данных.

В качестве алгоритма обучения используется оптимизированная версия CART. В качестве индекса неоднородности используется индекс Джини. Максимальная глубина дерева ограничена до 5 уровней.

Как показывает рисунок 5.2 дерево принятия решений смогло выделить все классы, что видно по главной диагонали матрицы ошибок. Лучше всего распознались следующие жанры: классическая музыка – 61 %, джаз – 56 %, поп – 50 % и метал – 58 %. Хуже всего: диско – 29 %, рок – 25 %.

Оценка перекрёстной проверки с десятью разбиениями – 42 % со средней квадратичным отклонением 4 %.

При классификации всего трека использовалась интегрированная оценка по всем фрагментам. Класс трека определялся наиболее часто встречаемым предсказанным классом его фрагментов. Это улучшило результаты

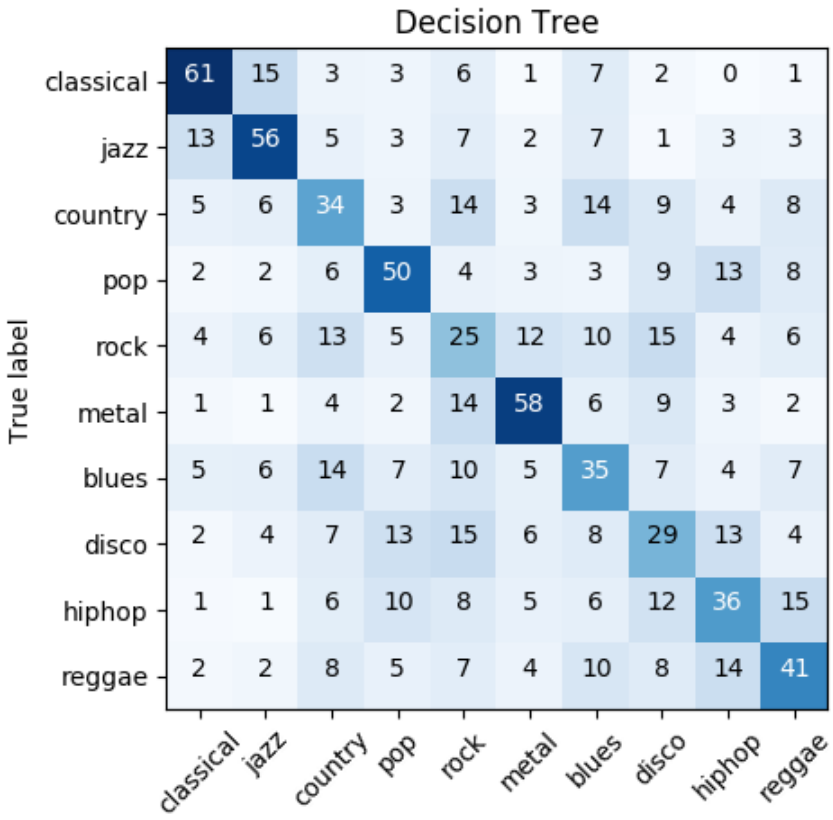

Рисунок 5.2 – Матрица ошибок для фрагмента трека полученная с помощь дерева принятия решения

классификации, как можно видеть на рисунке 5.3.

5.3 Метод опорных векторов

Метод опорных векторов – набор схожих алгоритмов обучения с учителем, использующихся для задач классификации и регрессионного анализа. Принадлежит семейству линейных классификаторов и может также рассматриваться как специальный случай регуляризации по Тихонову. Особым свойством метода опорных векторов является непрерывное уменьшение эмпирической ошибки классификации и увеличение зазора, поэтому метод также известен как метод классификатора с максимальным зазором.

Основная идея метода – перевод исходных векторов в пространство более высокой размерности и поиск разделяющей гиперплоскости с максимальным зазором в этом пространстве. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

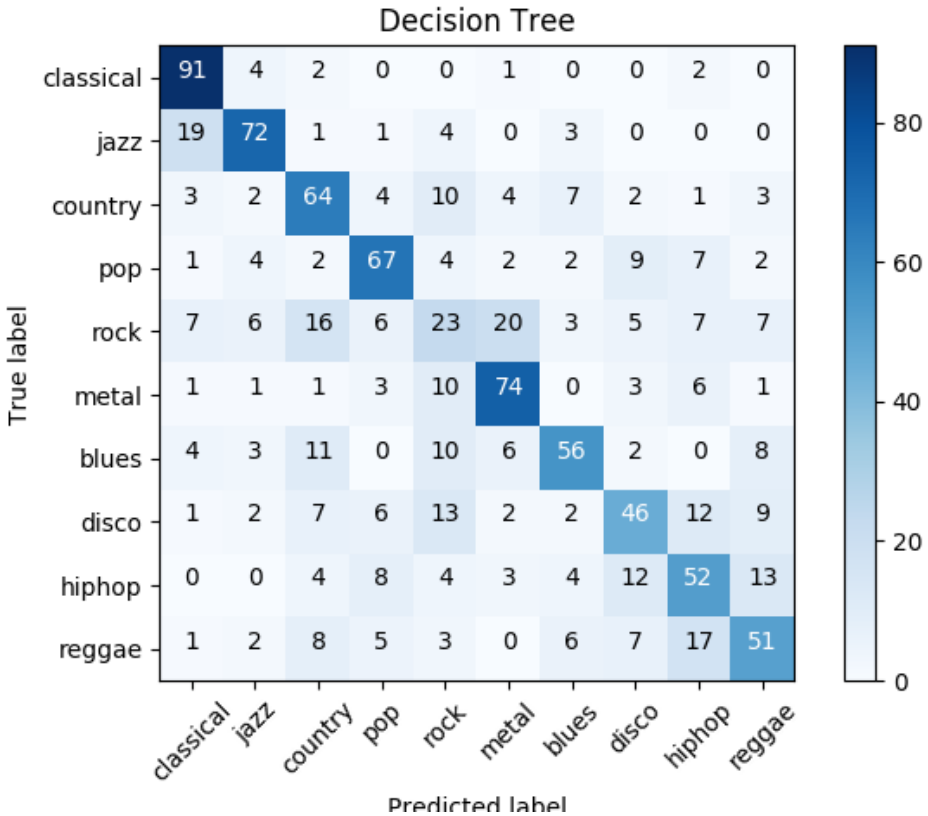

Рисунок 5.3 – Матрица ошибок для всего трека полученная с помощь дерева принятия решения

Как показывает рисунок 5.4 метод опорных векторов отлично справляется с задачи жанровой классификации.

Оценка перекрёстной проверки с десятью разбиениями – 66,3 % со средне квадратичным отклонением 0,23 %.

При классификации всего трека (см рисунок 5.5) обобщающая способность метода составила 76 %.

5.4 Наивный баесовский классификатор

Наивный байесовский классификатор – простой вероятностный классификатор, основанный на применении Теоремы Байеса со строгими (наивными) предположениями о независимости.

В зависимости от точной природы вероятностной модели, наивные байесовские классификаторы могут обучаться очень эффективно. Во многих практических приложениях для оценки параметров для наивных байесовых моделей используют метод максимального правдоподобия; другими словами, можно работать с наивной байесовской моделью, не веря в байесовскую вероятность и не используя байесовские методы.

Несмотря на наивный вид и, несомненно, очень упрощенные условия, наивные байесовские классификаторы часто работают намного лучше во

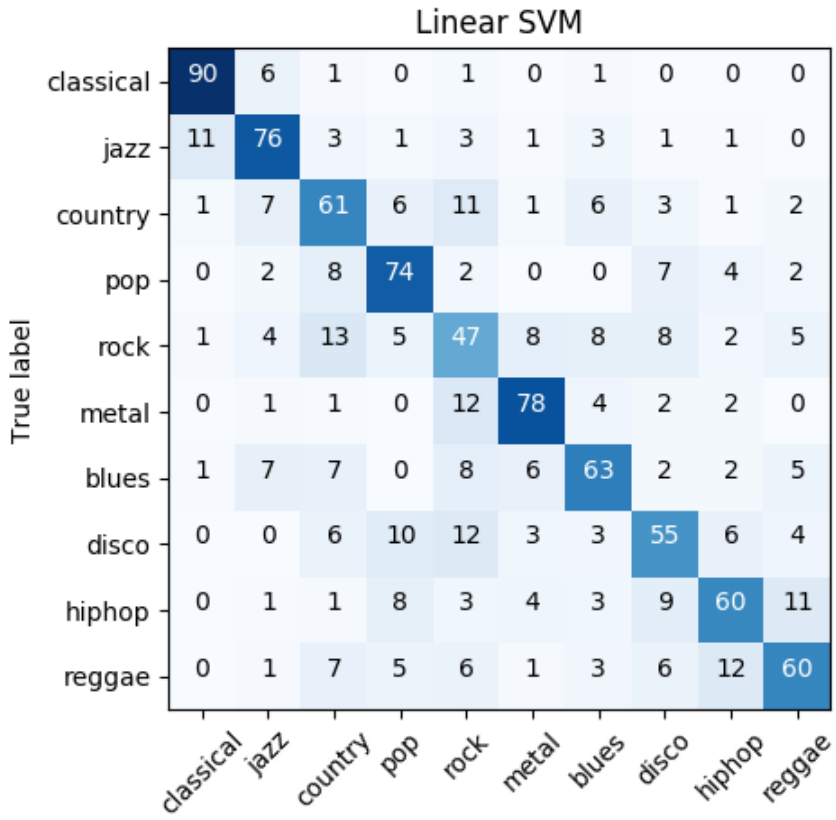

Рисунок 5.4 – Матрица ошибок для фрагмента трека полученная с помощь метода опорных векторов

многих сложных жизненных ситуациях.

Достоинством наивного байесовского классификатора является малое количество данных для обучения, необходимых для оценки параметров, требуемых для классификации.

Как показывает рисунок 5.6 наивный баесовский классификатор смог выделить все классы, что видно по главной диагонали матрицы ошибок. Лучше всего распознались следующие жанры: классическая музыка – 81 %, металл – 77 %, диско – 59 % . Хуже всего: хипхоп – 10 % и рок – 11 %. Рок чаще всего идентифицируется как кантри, металл или диско.

Оценка перекрёстной проверки с десятью разбиениями – 47,8 % со средне квадратичным отклонением 3 %.

При классификации всего трека (см рисунок 5.7) обобщающая способность метода составила 52 %.

5.5 Метод ближайших соседей

Метод k ближайших соседей – метрический алгоритм для автоматической классификации объектов. Основным принципом метода ближайших соседей является то, что объект присваивается тому классу, который является наиболее распространённым среди соседей данного элемента.

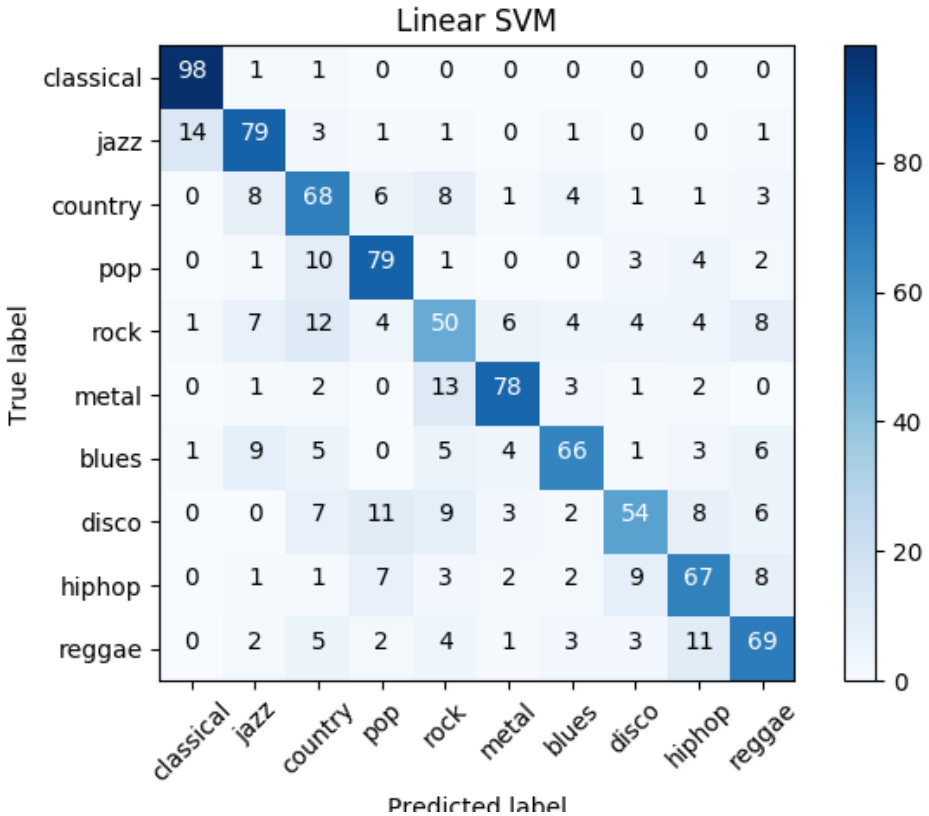

Рисунок 5.5 – Матрица ошибок для всего трека полученная с помощь метода опорных векторов

Соседи берутся исходя из множества объектов, классы которых уже известны, и, исходя из ключевого для данного метода значения k высчитывается, какой класс наиболее многочислен среди них. Каждый объект имеет конечное количество атрибутов (размерностей).

Предполагается, что существует определенный набор объектов с уже имеющейся классификацией.

Как показывает рисунок 5.8 метод ближайших соседей смог выделить все классы, что видно по главной диагонали матрицы ошибок. Лучше всего распознались следующие жанры: классическая музыка – 86 %, металл – 74 %, поп – 64 % . Хуже всего: блюз – 38 % и хипхоп – 28 %. Также заметна сильная погрешность при классификации джаза – 32 % фрагментов этого жанра было отнесено к классической музыке. Изменение числа соседей приводило к лучшей классификации металла, классической и поп музыки.

Оценка перекрёстной проверки с десятью разбиениями – 50,6 % со средне квадратичным отклонением 2,5 %.

При классификации всего трека (см рисунок 5.9) обобщающая способность метода составила 69 %.

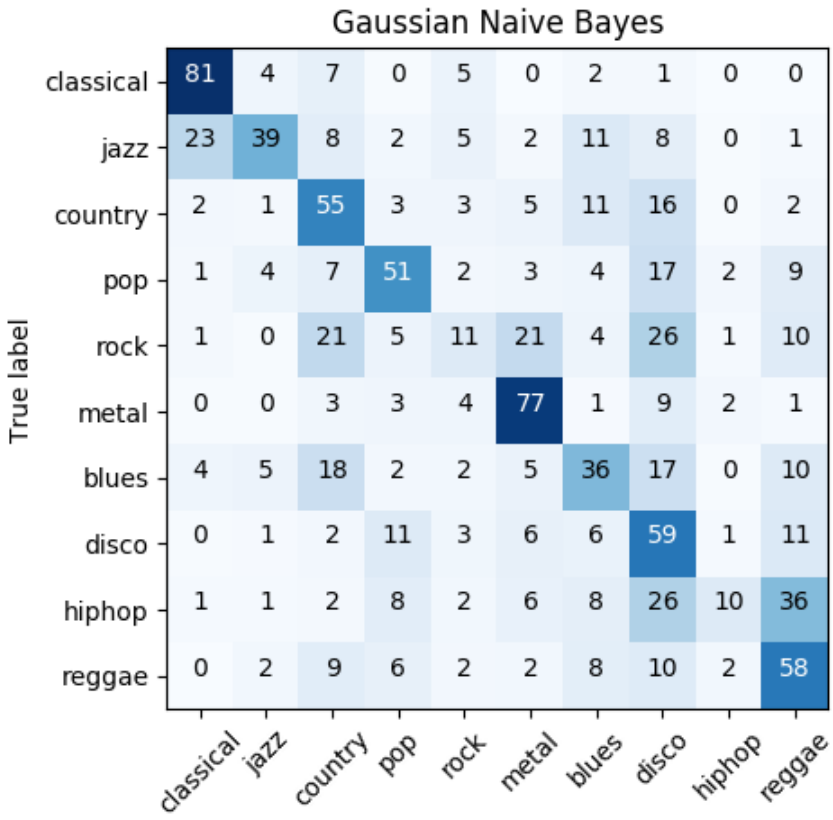

Рисунок 5.6 – Матрица ошибок для фрагмента трека полученная с помощь наивного баесовского классификатора

5.6 Многослойный персептрон

Многослойными персептронами называют нейронные сети прямого распространения. Входной сигнал в таких сетях распространяется в прямом направлении, от слоя к слою. Многослойный персептрон в общем представлении состоит из следующих элементов:

- множества входных узлов, которые образуют входной слой;
- одного или нескольких скрытых слоев вычислительных нейронов;
- одного выходного слоя нейронов.

Многослойный персептрон представляет собой обобщение однослойного персептрона Розенблатта.

В данной задачи использовался один скрытый слой с 1000 нейронами. В качестве функции активации использовалась положительно полулинейная функция.

Как показывает рисунок 5.10 многослойный перцептрон смог выделить все классы, что видно по главной диагонали матрицы ошибок. Данный метод классификации показал наилучшие результаты. Качество распознования по всем классам не ниже 50 процентов.

Оценка перекрёстной проверки с десятью разбиениями – 71 % со средне квадратичным отклонением 2,5 %

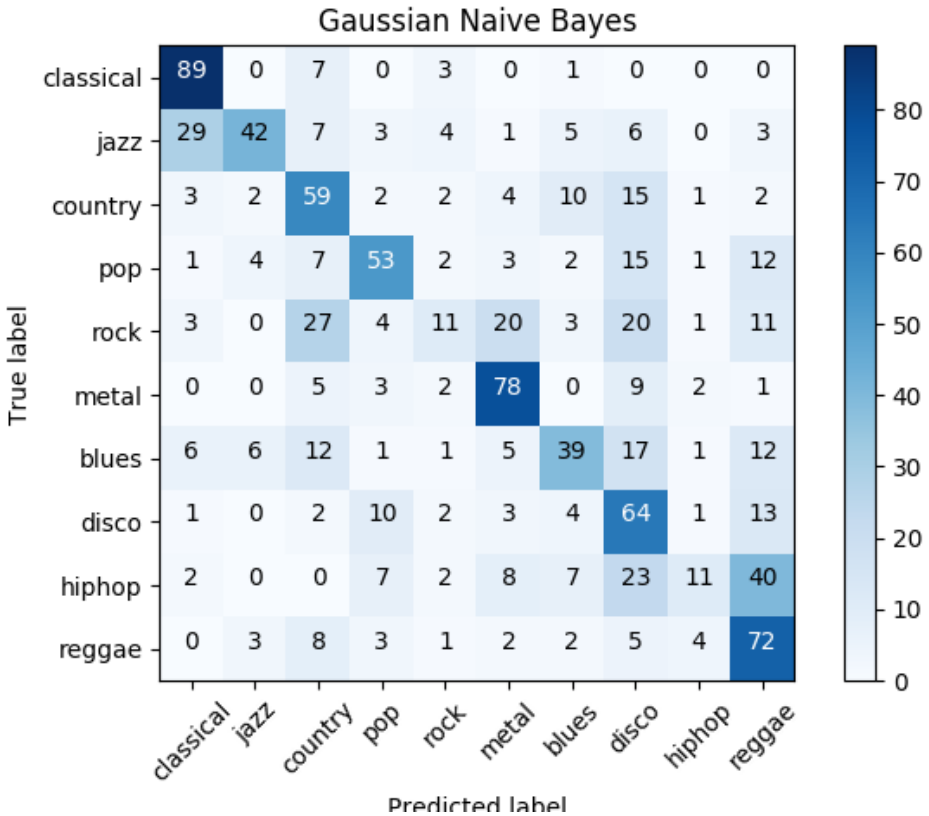

Рисунок 5.7 – Матрица ошибок для всего трека полученная с помощь наивного баесовского классификатора

При классификации всего трека (см рисунок 5.11) обобщающая способность метода составила 81 %. Качество распознования увеличилось по всем классам, кроме рока.

5.7 Квадратичный дискриминант

Квадратичный дискриминант - это вариант Байесовского классификатора, который основывается на двух дополнительных допущениях, касающихся вероятностных свойств выборки, а именно - независимость выборки и ее нормальность. Нормальное (гауссово) распределение широко используется по причине вычислительного удобства и адекватности во многих случаях.

Как показывает рисунок 5.12 при достаточно хорошей оценки перекрёстной проверки видно, что сам метод классификации не подходит, так как 20 % ошибок на других жанрах он классифицирует как блюз.

При классификации всего трека (см рисунок 5.13) обобщающая способность метода составила 51 %.

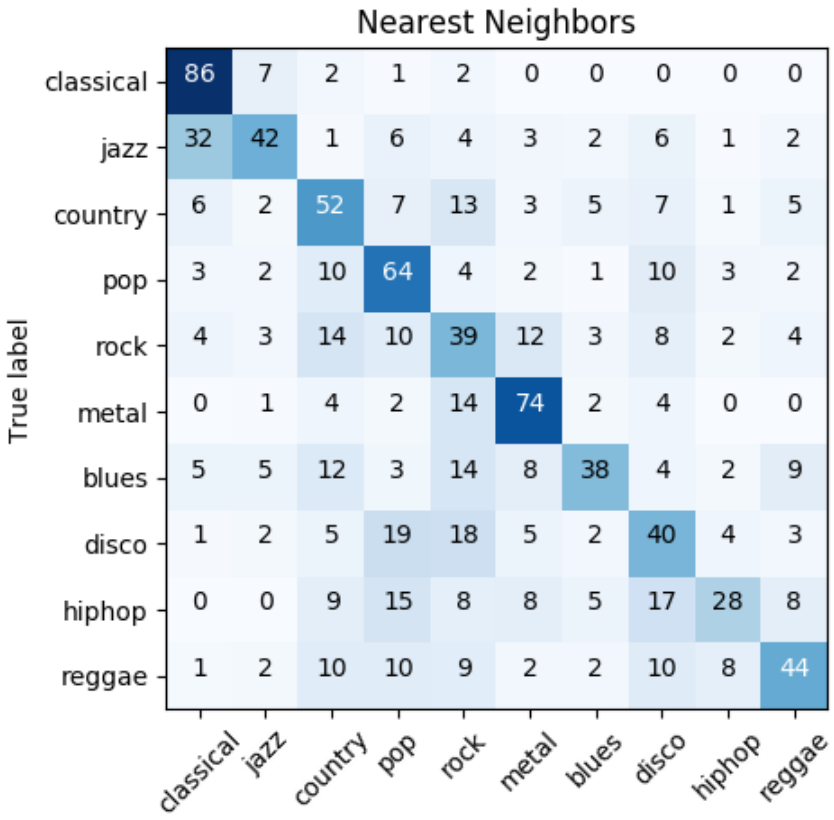

Рисунок 5.8 – Матрица ошибок для фрагмента трека полученная с помощь метода k ближайших соседей

5.8 Случайный лес

Случайный лес – алгоритм машинного обучения, предложенный, заключающийся в использовании комитета (ансамбля) решающих деревьев. Классификация объектов проводится путём голосования: каждое дерево комитета относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев.

Оптимальное число деревьев подбирается таким образом, чтобы минимизировать ошибку классификатора на тестовой выборке. В случае её отсутствия, минимизируется оценка ошибки out-of-bag: доля примеров обучающей выборки, неправильно классифицируемых комитетом, если не учитывать голоса деревьев на примерах, входящих в их собственную обучающую подвыборку.

Оценка перекрёстной проверки с десятью разбиениями – 39 % со средне квадратичным отклонением 3 %

При классификации всего трека (см рисунок 5.13) обобщающая способность метода составила 54 %.

В результате даже самые элементарные алгоритмы классификации показали свою эффективность, что показывает, что выделенные информационные образы значимы и могут быть использованны в системах рекомендации

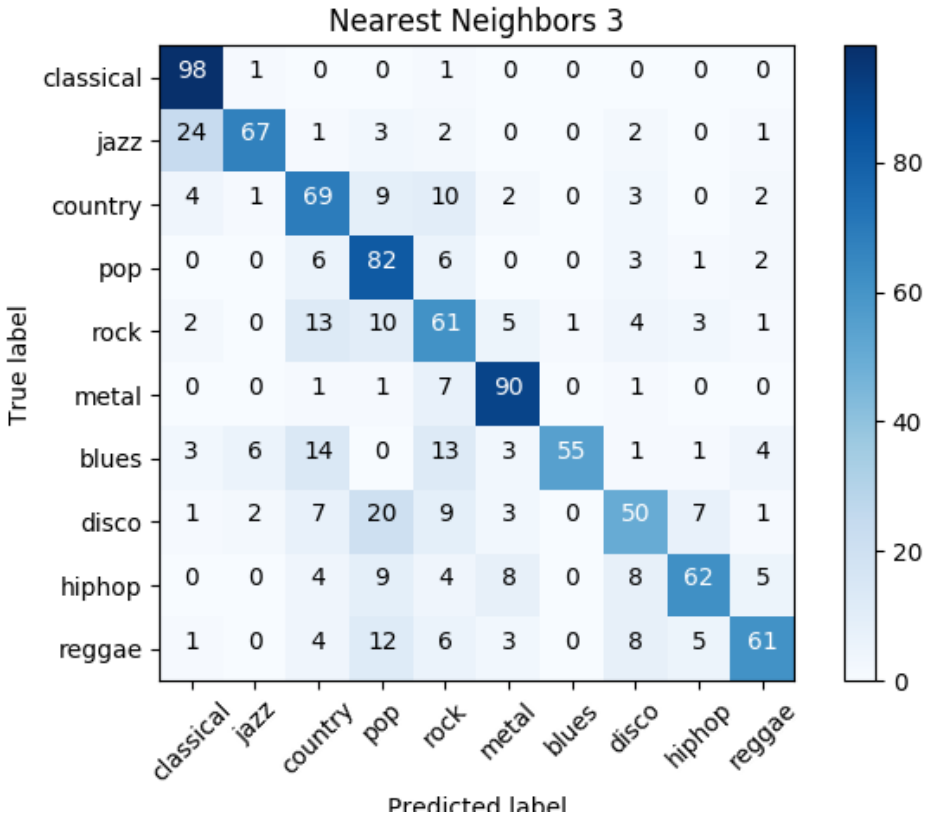

Рисунок 5.9 – Матрица ошибок для всего трека полученная с помощь метода k ближайших соседей

музыки. Лучше всего распозналась музыка в жанре поп, металл и классическая музыка. Хуже всего рок. Все результаты сведены в таблицу 5.1.

Таблица 5.1 – Таблица результатов классификации

Методы классификации	фрагмент 5 сек. перекрытие 0.1 %	Трек, на основе фрагмента 5 сек. перекрытие 0.5 %
AdaBoost с деревьями принятия решений	28,2	44
Дерево принятия решений	42,4	59
Метод опорных векторов	66,3	76
Наивный баесовский классификатор	47,8	52
Метод ближайших соседей	50,6	69
Нейронная сеть прямого распространения	71,0	81
Квадратичный дискриминант	47,9	51
Случайный лес	39,0	52

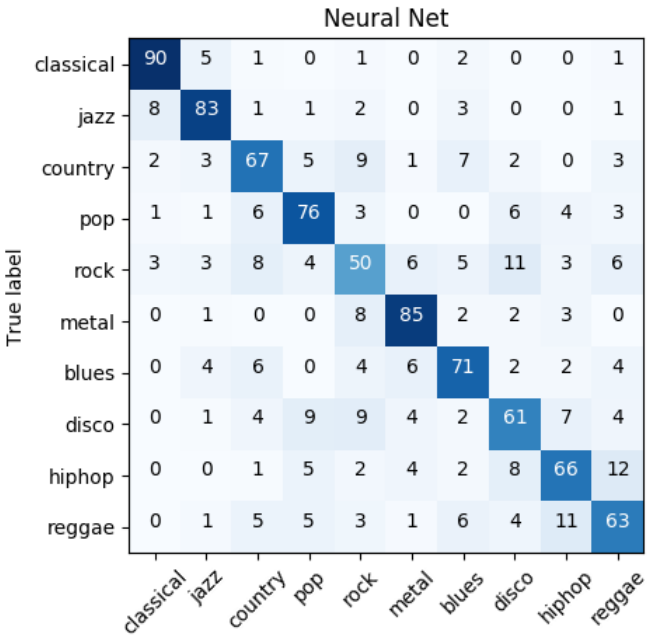

Рисунок 5.10 – Матрица ошибок для фрагмента трека полученная с помощь многослойного перцептрона

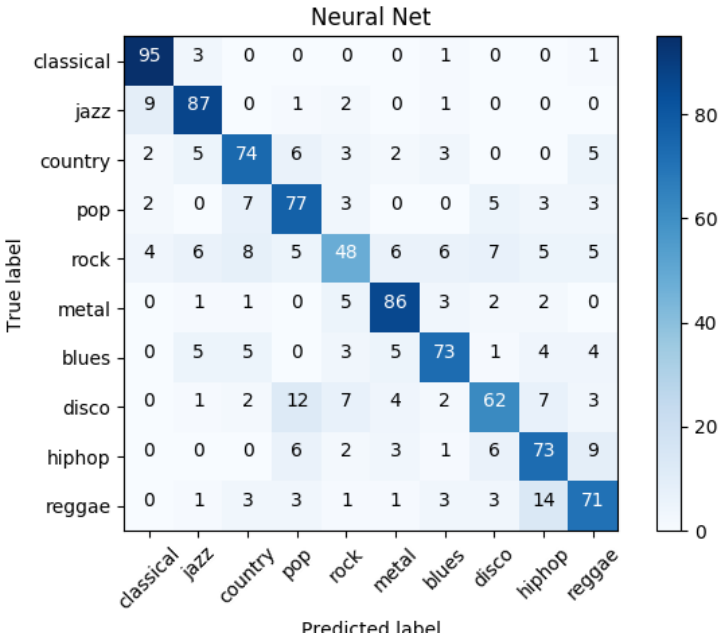

Рисунок 5.11 – Матрица ошибок для всего трека полученная с помощь многослойного перцептрона

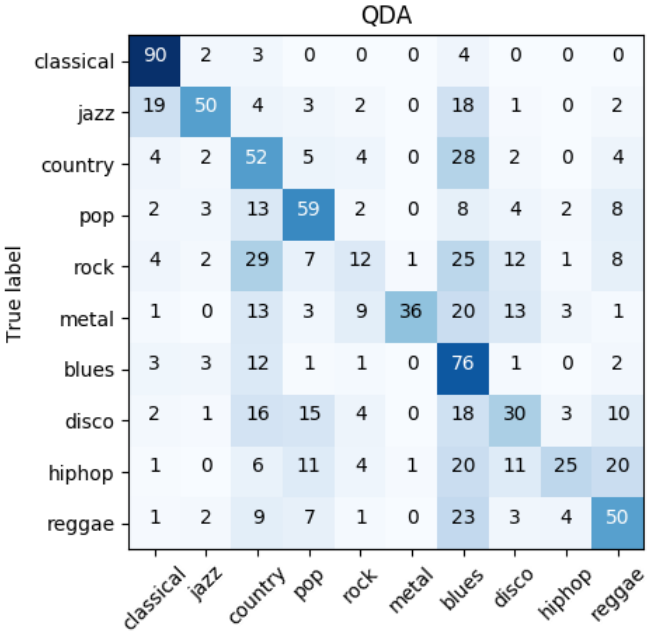

Рисунок 5.12 – Матрица ошибок для фрагмента трека полученная с помощь QDA

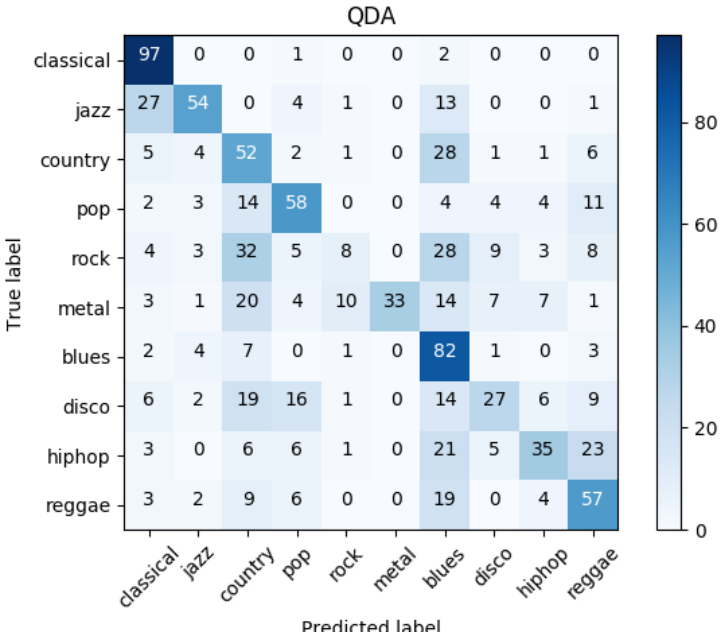

Рисунок 5.13 – Матрица ошибок для всего трека полученная с помощь QDA

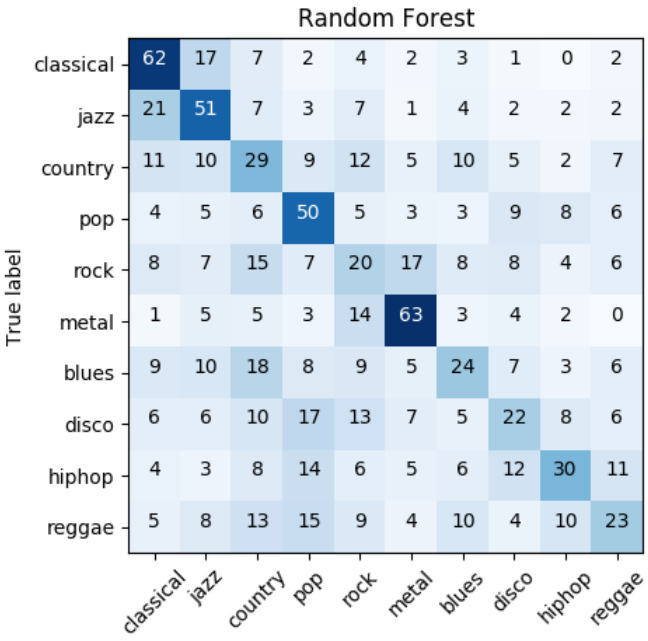

Рисунок 5.14 – Матрица ошибок для фрагмента полученная с случайного леса

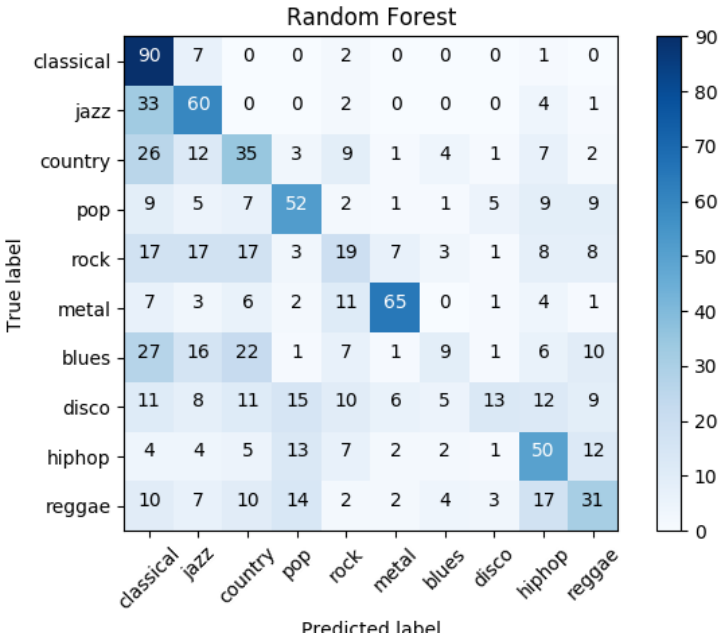

Рисунок 5.15 – Матрица ошибок для всего трека полученная с помощь QDA

6 РЕЗУЛЬТАТЫ ВИЗУАЛИЗАЦИИ

Для большей наглядности были решено визуализировать только те жанры, которые лучше всего выделяются всеми алгоритмами классификации. Из 10 жанров были выбраны следующие:

- классическая музыка;
- музыка в жанре металл;
- музыка в жанре поп.

Визуализация проводилась над фрагментами музыкального произведения.

6.1 Метод главных компонент

Метод главных компонент – один из основных способов уменьшить размерность данных, потеряв наименьшее количество информации. Применяется во многих областях, таких как распознавание образов, компьютерное зрение, сжатие данных и т. п. Вычисление главных компонент сводится к вычислению собственных векторов и собственных значений ковариационной матрицы исходных данных или к сингулярному разложению матрицы данных.

По двумерному отображению данных видно, что кластеры ярко выражены и линейно разделимы. Особой компактностью отличается музыка в жанре поп. Кластеры с классической музыкой и с музыкой в жанре металл, также ярко выражены.

Трёхмерное отображение показывает те же результаты, что и двумерная. В обоих проекциях виден малый зазор между классами.

6.2 t-SNE

t-SNE (t-distributed stochastic neighbor embedding, стохастическое вложение соседей с распределением Стьюдента) – алгоритм уменьшения размерности.

Точка данных (data point) – это точка x_i в исходном пространстве данных R^D (data space), где D – размерность (dimensionality) пространства данных.

Точка отображения (map point) – это точка y_i в пространстве отображения R^2 (map space). Это пространство будет содержать целевое представление набора данных.

Более конкретно, если две точки данных расположены близко друг к другу, необходилм, чтобы две соответствующие точки отображения также располагались близко друг к другу. Пусть $|x_i - x_j|$ – евклидово расстояние между двумя точками данных, а $|y_i - y_j|$ – расстояние между точками отображения. Сначала определим условное сходство (conditional similarity) для

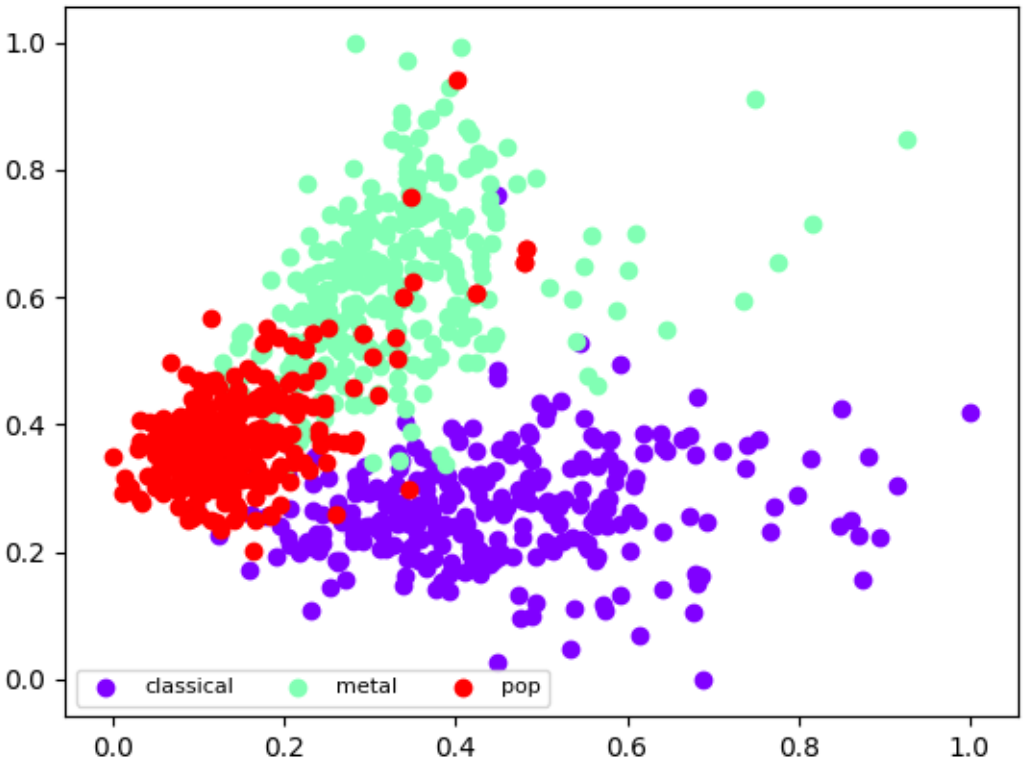

Рисунок 6.1 – Отображения пространства признаков музыкальных треков в двухмерное простарнство алгоритмом PCA.

двух точек данных:

$$p_{j|i} = \frac{exp(\frac{-|x_i - x_j|^2}{2\sigma_i^2})}{\sum\limits_{k \neq i} exp(\frac{-|x_i - x_j|^2}{2\sigma_i^2})} \tag{6.1}$$

Это выражение показывает, насколько точка x_j близка к x_i, при гауссовом распределении вокруг x_i с заданной дисперсией σ_i^2. Дисперсия различна для каждой точки. Она выбирается таким образом, чтобы точки, расположенные в областях с большой плотностью, имели меньшую дисперсию, чем точки, расположенные в областях с малой плотностью.

Теперь определим сходство, как симметричный вариант условного сходства:

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{6.2}$$

Получаем матрицу сходства (similarity matrix) для исходного набора данных.

Также получаем матрицу сходства для точек отображения.

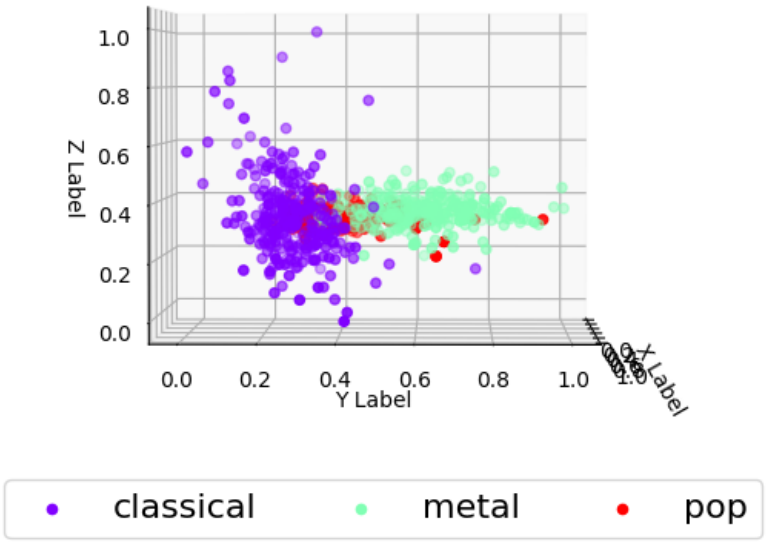

Рисунок 6.2 – Отображения пространства признаков музыкальных треков в трёхмерное пространство алгоритмом PCA. Первая проекция.

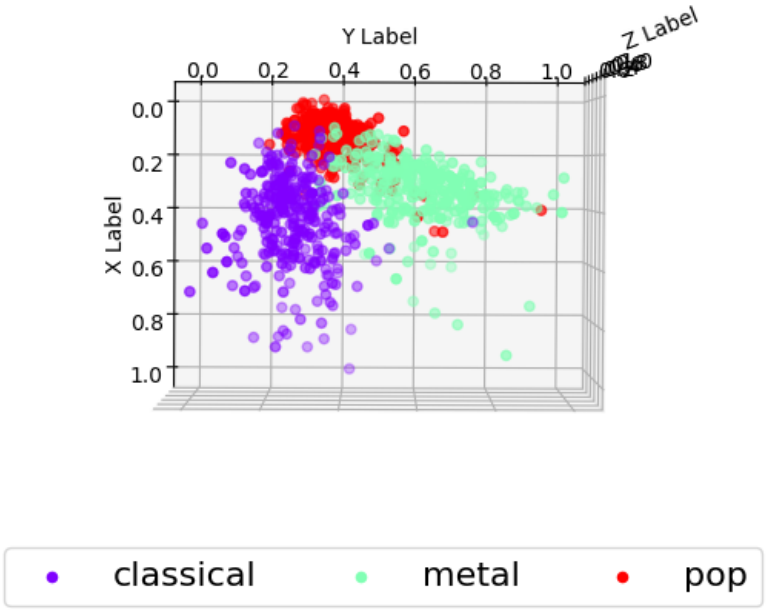

Рисунок 6.3 – Отображения пространства признаков музыкальных треков в трёхмерное пространство алгоритмом PCA. Вторая проекция.

$$q_{j|i} = \frac{f(|x_i - x_j|)}{\sum\limits_{k \neq i} f(|x_i - x_j|)} \tag{6.3}$$

$$f(z) = \frac{1}{1 + z^2} \tag{6.4}$$

Для меры сходства используется расстояния Кульбака-Лейблера между двумя распределениями (p_{ij}) и (q_{ij}):

$$KL(P||Q) = \sum_{i,j} p_{ij} log \frac{p_{ij}}{q_{ij}} \tag{6.5}$$

Данная формула выражает расстояние между двумя матрицами сходства.

Чтобы минимизировать эту величину, применим градиентный спуск. Градиент может быть вычислен аналитически:

$$\frac{\partial KL(P||Q)}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij}) g(|x_i - x_j|) u_{ij} \tag{6.6}$$

Здесь u_{ij} – единичный вектор, идущий от y_j к y_i. Этот градиент выражает сумму всех сил, приложенных к точке отображения i.

$$g(z) = \frac{1}{1 + z^2} \tag{6.7}$$

Теперь объясним, почему для точек отображения было выбрано распределение Стьюдента, в то время как для точек данных применяется нормальное распределение. Известно, что объем N-мерного шара радиуса r пропорционален r^N. При больших N, если выбирать случайные точки в шаре, большинство точек будет располагаться около поверхности, и очень небольшое количество – около центра.

При уменьшении размерности набора данных, если использовать гауссово распределение для точек данных и точек отображения, мы получим дисбаланс в распределении расстояний для соседей точек. Это объясняется тем, что распределение расстояний существенно отличается для пространства большой размерности и для пространства малой размерности. Тем не менее, алгоритм пытается воспроизвести одинаковые расстояния в обоих пространствах. Этот дисбаланс создает избыток сил притяжения, что ино-

гда приводит к неудачному отображению.

Алгоритм t-SNE решает эту проблему, используя распределение Стьюдента с одной степенью свободы (или распределение Коши) для точек отображения. В отличие от гауссова распределения, это распределение имеет значительно более «тяжелый» хвост, что позволяет компенсировать дисбаланс. Для данного сходства между двумя точками данных, две соответствующие точки отображения должны находиться намного дальше друг от друга, чтобы их сходство соответствовало сходству точек данных. Это можно увидеть на следующем графике.

Использование этого распределения обеспечивает более эффективную визуализацию данных, при которой группы точек более отчетливо отделены друг от друга.

Алгоритм t-SNE обеспечивает эффективный метод визуализации сложных наборов данных. Он успешно обнаруживает скрытые структуры в данных, демонстрирует группы и компенсирует нелинейные отклонения по измерениям.

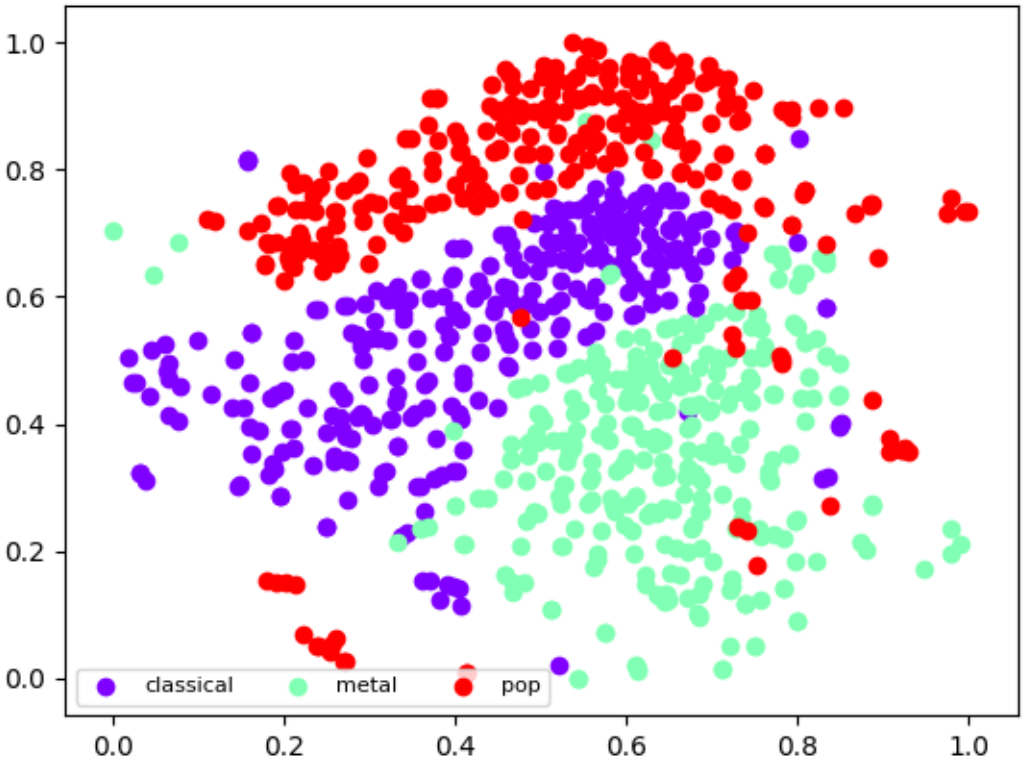

Рисунок 6.4 – Отображения пространства признаков музыкальных треков в двухмерное простарнство алгоритмом t-SNE.

Визуализация также показала, что выделенные информационные образы значимы и на их основе можно делать рекомендательный сервис.

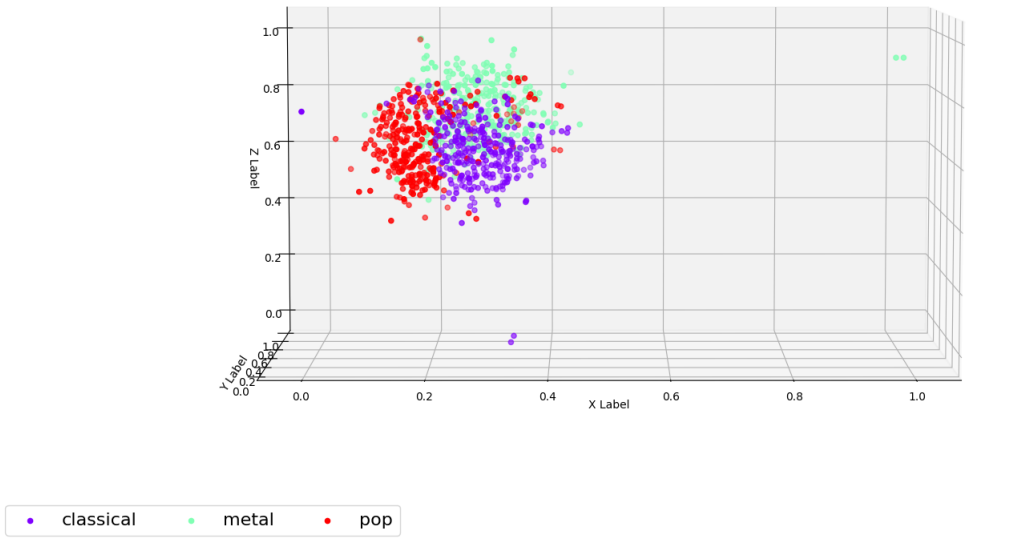

Рисунок 6.5 – Отображения пространства признаков музыкальных треков в трёхмерное пространство алгоритмом t-SNE. Первая проекция.

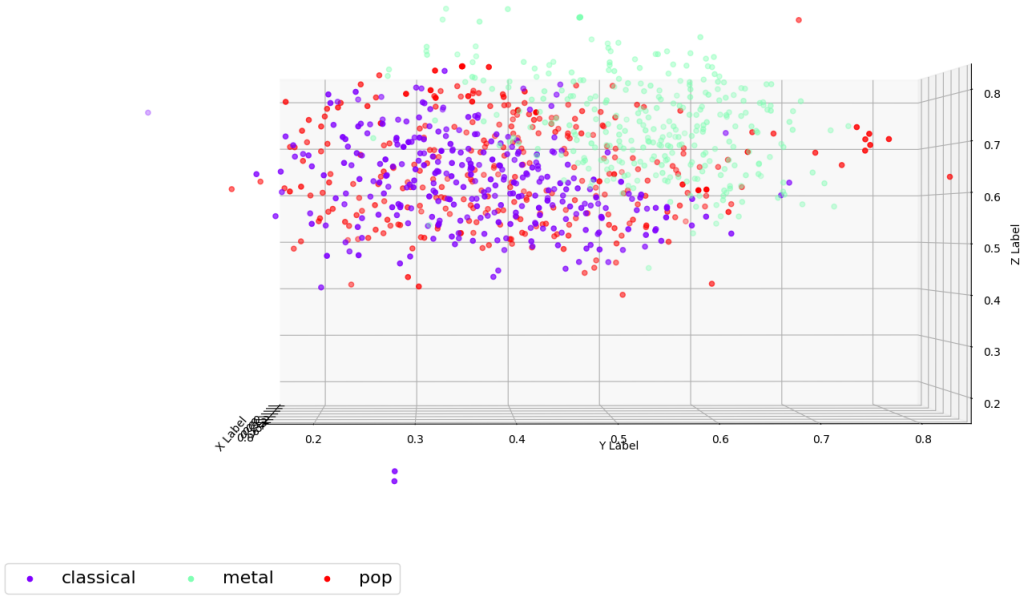

Рисунок 6.6 – Отображения пространства признаков музыкальных треков в трёхмерное пространство алгоритмом t-SNE. Вторая проекция.

7 ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Тестирование программного обеспечения - это анализ, проводимый с целью предоставления заинтересованным сторонам информации о качестве тестируемого продукта или услуги. Тестирование программного обеспечения также может обеспечить объективное независимое представление программного обеспечения, которое позволит бизнесу оценить и понять риски внедрения программного обеспечения. Методы тестирования включают в себя процесс выполнения программы или приложения с целью обнаружения ошибок программного обеспечения (ошибок или других дефектов) и проверки пригодности программного продукта для использования.

Тестирование программного обеспечения включает в себя выполнение программного компонента или системного компонента для оценки одного или нескольких интересующих свойств. Как правило, эти свойства указывают, в какой степени испытываемый компонент или система:

- отвечает требованиям, которые руководствовались его дизайном и разработкой;
- правильно реагирует на все виды входных данных;
- работает в течение приемлемого времени;
- является достаточно удобной для использования;
- может быть установлена и запущена в своих рабочих средах;
- достигает общего результата, к которому стремятся его заинтересованные стороны.

Поскольку количество возможных тестов для даже простых программных компонентов практически бесконечно, всё тестирование программного обеспечения использует некоторую стратегию для выбора тестов, которые являются выполнимыми для доступного времени и ресурсов. В результате тестирование программного обеспечения обычно (но не исключительно) пытаются выполнить программу или приложение с целью обнаружения ошибок программного обеспечения (ошибок или других дефектов). Задача тестирования – это итеративный процесс, когда исправление одной ошибки может привести к нахождению другой ошибки или даже создать новую.

Тестирование программного обеспечения может обеспечить объективную независимую информацию о качестве программного обеспечения и риск его отказа пользователям или спонсорам.

Общий подход к разработке программного обеспечения часто определяет, когда и как проводится тестирование. Например, при поэтапном процессе разработки большинство тестов выполняется после того, как системные требования определены, а затем реализованы в тестируемых программах. Напротив, в рамках подхода (agile) требования, программирование и тестирование часто выполняются одновременно.

В данном проекте для повышения качества разрабатываемой продукции тестирование совершалось по принципу гибкой (agile) модели:

- люди и взаимодействие важнее процессов и инструментов;
- работающий продукт важнее исчерпывающей документации;
- сотрудничество с заказчиком важнее согласования изначальных условий контракта;
- готовность к изменениям важнее следования первоначальному плану разработки.

Благодаря концепциям, вложенным в основу гибкой модели тестирования, в результате их применения в проекте наблюдалось максимальная адаптации процесса разработки программного обеспечения к мгновенным изменениям требований заказчика.

В целом тестирование было разделено на два этапа:

- первый этап – тестирование отдельных модулей программы в процессе написания программного кода – т.н. модульное тестрование;
- второй этап – тестирование программного продукта целиком – т.н. интеграционное тестирование.

Оба этапа являются достаточно важными, так как работа каждого из них влияет на процесс проверки качества. Например, без модульного тестирования, при анализе работы программы в целом, будет происходить достаточное количество сбоев, выявить и локализовать которые может оказаться достаточно сложным заданием, в то время как при анализе работы одного модуля неисправность оказывается достаточно очевидной. И обратный случай, работоспособность каждого компонента в отдельности не гарантирует корректное поведение всей программы в целом.

Для модульного тестирования был выбран входищий в стандартную библиотеку фреймворк для модульного тестирования под названием unittest. Unittest поддерживает автоматизацию тестов, разделение кода установки и завершения тестов, агрегацию тестов в коллекции и независимость тестов от структуры отчетности. Модуль unittest предоставляет классы, которые позволяют легко поддерживать эти качества для набора тестов.

При построении модульных тестов программы используется так называемый подход «AAA», что означает «Arrange, Act, Assert». Сущность подхода состоит в том, что модульный тест заключает в себе три фундаментальных действия (некоторые из которых могут быть разделены между различными тестами одного набора, покрывающими один компонент).

- часть «Arrange» подразумевает установку компонента в требуемое исходное состояние путем инициализации компонента нужными значениями, вызовами мутирующих внутреннее состояние методов и т.д;
- часть «Act» подразумевает выполнение тестируемого действия. Как правило заключается в вызове тестируемого метода или группы методов;
- часть «Assert» заключается в проверке результатов вызова тестируемого действия, соответствия внутреннего состояния системы ожидаемому по завершению вызова, факте наличия либо отсутствия необработанных ис-

ключений и т.д;

```
import unittest
import HelloWorld
 # TestModule test suite
class TestStringMethods(unittest.TestCase):
  # Initialization - Arrange
  hello_world = HelloWorld()

  # Unit test
   def test_upper(self):
      # Act & Assert
      self.assertEqual('foo'.upper(), 'FOO')

  # Unit test
   def test_split(self):
      # Arrange
      s = self.hello_world.get_name()
       # Act & Assert
      self.assertEqual(s.split(), ['hello', 'world'])


if __name__ == '__main__':
   unittest.main()

   ...
```

Следуя вышепреведенному образцу был реализован набор тестов, покрывающий различные классы которые наследуются от `FeatureExtractorMode`.

Одной из сложностей модульного тестирования является написание такого модуля, чьи зависимости от других модулей системы максимально абстрагированы и отделены друг от друга. Иногда ввиду наличия сложных связей между разными модулями, их невозможно тестировать по отдельности (например, в случаях когда существует зависимость одного модуля от другого на этапе компиляции, такая как зависимость в конструкторе). В таких случаях принято использовать так называемые mock-объекты (ложные объекты), реализующие тот же интерфейс, что и модуль, от которого зависит тестируемый, но реализующий логику данного интерфейса достаточно просто, чтобы с гораздо большей долей вероятности не иметь неисправностей в своей реализации.

Произведя тестирование командой `python unittest -v unittests` получаем следующий вывод результатов тестирования:

```
python -m unittest -v unittests
```

```
test_energy (unittests.TestFeatureExtractor) ... ok
test_first_order_autocorrelation (unittests.
  TestFeatureExtractor) ... ok
test_zero_crossing_rate (unittests.
  TestFeatureExtractor) ... ok
test_spectal_centroid (unittests.TestFeatureExtractor)
   ... ok
test_spectral_smoothness (unittests.
  TestFeatureExtractor) ... ok
test_spectral_spread (unittests.TestFeatureExtractor)
  ... ok
test_spectral_dissymmetry (unittests.
  TestFeatureExtractor) ... ok
test_linear_regression (unittests.TestFeatureExtractor
  ) ... ok
test_rolloff (unittests.TestFeatureExtractor) ... ok
test_sfm (unittests.TestFeatureExtractor) ... ok
test_scf (unittests.TestFeatureExtractor) ... ok


----------------------------------------------------------------

Ran 11 tests in 0.04s

OK
```

Для тестирования классов которые наследуются от `FeatureExtractorModel` в качестве входных данных использовался кусок синусоиды и её спектр.

```
import unittest


class TestFeatureExtractor(unittest.TestCase):

   def setUp(self):
     # create seq
      self.test_data = np.linspace(-np.pi * 100,
                                   np.pi * 100,
                                   500)
      # get spectral data
      self.test_data_spectre = scipy.fft(self.
        test_data)
```

```
    # test class Energy
    def test_energy(self):
        energy = Energy()
        # get seq energy
        result = energy.get(self.seq)
        # energy must be greater than zero
        self.assertGreater(result, 0)

    # test class ZeroCrossingRate
    def test_zero_crossing_rate(self):
        zcr = ZeroCrossingRate()
        # get zero crossing rate
        rate = zcr.get(np.sin(self.test_data))
        real_rate = 200.0 / len(self.test_data)
        self.assertGreater(rate, real_rate)

        ...
```

Модульное тестирование наиболее эффективно, когда оно является неотъемлемой частью рабочего процесса разработки программного обеспечения. После написания функции или другого блока кода приложения создаются модульные тесты, которые проверяют поведение кода в ответ на стандартные, граничные и некорректные случаи ввода данных, также проверяются любые явные или предполагаемые допущения, сделанные кодом. В практике разработки программного обеспечения, известной как разработка, управляемая тестом, создается модуль тестов перед написанием кода, поэтому модульные тесты используются в качестве технической документации и спецификации функциональности.

После того, как модульные тесты были закончены для большей части функциональности и реализованные методы прошли все необходимые пункты тестирования, началось функциональное тестирование.

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения исходным функциональным требованиям заказчика. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

В зависимости от степени доступа к коду системы можно выделить два типа функциональных испытаний:

- тестирование black box (черный ящик) – проведение функционального тестирования без доступа к коду системы;
- тестирование white box (белый ящик) – функциональное тестирование с доступом к внутренней структуре и коду приложения.

Таблица 7.1 – Тестирование программы

Модуль	Содержание теста	Ожидаемый результат	Тест пройден
WavModule	Конвертация MP3 в WAV	WAV-файл лежит в той же директории, что и MP3	да
WavModule	Считываение WAV-файла	Файл считывается в массив	да
Preprocessing Module	Разбивка трека на фрагменты	Трек разбивается на заданное количество фрагментов	да
Spectral Transformer	Получение коррелограммы на основе фрагмента трека	Коррелограмма получена и сохранена в массив	да
Feature Processing	Вычисления МО и СКО	МО и СКО вычисленны и сохранены в вектор признаков	да
Genre Classification Module	Запуск каждого метода классификации в своём процессе	Выполнения процесса классификации на всех ядрах	да
Visualize DataModule	Сохранения графика в pdf	Файл с графиком находится в директории	да

Тестирование black box проводится без знания внутренних механизмов работы системы и опирается на внешние проявления ее работы. При этом тестировании проверяется поведение программного обеспечения при различных входных данных и внутреннем состоянии систем.

Основной информацией для создания тест-кейсов выступает документация (особенно — требования (requirements-based testing). В случае тестирования white box создаются тест-кейсы, основанные преимущественно на коде системы программного обеспечения. Также существует расширенный тип black-box тестирования, включающего в себя изучение кода, – так называемый grey box (серый ящик). Концепция серого ящика заключается в комбинации методов белого ящика и чёрного ящика, состоящая в том, что к части кода и архитектуре доступ есть, а к части — нет.

Тестирование функциональности может проводиться в двух аспектах:

требования и бизнес-процессы. Тестирование в перспективе «требования» используют спецификацию функциональных требований к системе как основу для дизайна тестовых случаев (Test Cases). В этом случае необходимо сделать список того, что будет тестироваться, а что нет, приоритезировать требования на основе рисков, а на основе этого приоритезировать тестовые сценарии. Это позволяет сфокусироваться и не упустить при тестировании наиболее важный функционал.

Тестирование в перспективе «бизнес-процессы» используют знание этих самых бизнес-процессов, которые описывают сценарии ежедневного использования системы.

Тестирование данного программного продукта производилось несколькими пользователями на разных компьютерах:

- настольный компьютер Intel Core i7, 16 Гб RAM, Ubuntu 17.04;
- настольный компьютер Intel Core i5, 32 Гб RAM, Windows 10;
- ноутбук Intel Core i3, 4 Гб RAM, Windows 7.

В таблице 7.1 предоставлена информация о прохождении функциональных тестов данным программным обеспечением.

Как видно из таблицы, приложение хорошо справилось с тестами, что говорит о высокой работоспособности.

8 РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ

Модуль выделения информационных признаков из музыкального произведения представляет собой API на языке программирования Python, который решает задачи выделение спектральных, временных и иных признаков из музыкального трека, проверка значимости признаков путём использовании их в задаче жанровой классификации, визуализация данных алгоритмом t-SNE.

8.1 Требования к аппартному и программному обеспечению

Минимальные требования для полнооценного функционирования программного обеспечения:

- операционные системы Windows XP с пакетом обновления 2 +, Windows Vista, Windows 7, Windows 8, Windows 10, Mac OS X 10.6 или более поздней версии Ubuntu 10.04 +, Debian 6 +, OpenSuSE 11.3 +, Fedora Linux 14;
- установленный интерпретатор Python 2.7;
- пакетный менеджер pip;
- pip-пакет Scipy версии 19.0 +;
- pip-пакет scikit-learn версии 0.18.1 +;
- pip-пакет PeakUtils версии 1.1.0 +;
- pip-пакет scilits.talkbox версии 0.2.5+;
- pip-пакет pymongo;
- установленный MongoDB;
- процессор Intel Pentium 4 / Athlon 64 или более поздней версии с поддержкой SSE2;
- свободное место на жестком диске 350 Мб;
- оперативная память 512 Мб.

8.2 Руководство по установке и запуску программного средства

Для установки приложения необходимо запустить установщик main_module.e на любом жестком диске, в котором будет достаточно свободного пространства. Требования описаны выше. В результате этого создастся папка main_module, в которой будут находиться все скрипты нужные для функционирования модуля. Для использования модуля необходимо добавить скрипты в собственный проект и с помощью команды `import` подключить `MainModule` для выделения информационных образов, `GenreClassificationModule` для жанровой классификации и `VisualizeDataModule` для визуализации информационных образов.

8.3 Руководство по использованию программного модуля

Для начала работы с модулем необходимо создать объекты подмодулей с необходимыми параметрами.

Для считывания музыкального произведения необходимо создать объект класса `WavModule`, который не принимает параметров.

```
wav_module = WavModule()
```

Для создание модуля препроцессинга и нарезки необходимо создать объект класса `PreprocessingModule`, где в конструкторе указывать следующие параметры:

- `alpha` – коэффициент экспоненициального сглаживания;
- `cut_end` – процент трека, которое будет отсечено с конца трека;
- `cut_start` – процент трека, которое будет отсечено с конца трека;
- `overlap` – процент пересечения фрагментов трека;
- `frame_size_sec` – длина каждого фрагмента в секундах.

```
preprocessing_module = PreprocessingModule(
                            alpha=0.99,
                             cut_end=0.2,
                             cut_start=0.2,
                             overlap=0.1,
                             frame_size_sec=5)
```

Для создания модуля получения частотно-временного представления сигнала необходимо создать объект класса `SpectralTransformer`, где в конструкторе указывать слежующие параметры:

- `alpha` – коэффициент экспоненициального сглаживания;
- `level` – количество каскадов при вейвлет преобразовании Добеши;
- `rate` – процент пересечения фрагментов трека;
- `window` – массив содержащий окно.

```
window = signal.hamming(4096)
spectral_transformer = SpectralTransformer(
                            alpha=0.99,
                             level=4,
                             rate=16,
                             window=window)
```

Для создания модуля извленчения информационных образов необходим необходимо создать объект класса `FeatureProcessing`, где в конструкторе указыватеются следующие параметры:

- `models` – ассоциативный массив, где ключ является наследником класса `FeatureExtractorModel`, а значение - массив зависимых признаков ;
- `nceps` – количество мел-кепстральных коэффициентов;

```
models = {
        Energy: [],
        SpectralSmoothness: [],
        SpectralSpread: [SpectralCentroid],
        ...
}
feature_extractor = FeatureExtractor(
                                    models=models,
                                     nceps=24)
```

Для создания модуля обработки информационных образов необходимо создать объект класса `FeatureProcessing`, где в конструкторе указыватеются следующие параметры:

- `with_mean` – добавление в информационный вектор МО спектральных признаков и мел-кепстральных коэффициентов;
- `with_std` – добавление в информационный вектор СКО спектральных признаков и мел-кепстральных коэффициентов;
- `with_kurtosis` – добавление в информационный вектор коэффициента эксесса спектральных признаков и мел-кепстральных коэффициентов;
- `with_skew` – добавление в информационный вектор коэффициента склонения спектральных признаков и мел-кепстральных коэффициентов;

```
feature_processing=FeatureProcessing(
                            with_mean=True,
                            with_std=True
                            with_kurtosis=False,
                              with_skew=True)
```

Далее для созданные объекты отправляются в конструктор класса `MainModule`.

```
main_module = MainModule(
                      wav_module,
                      preprocessing_module,
                      spectral_transformer,
                      feature_extractor,
                      feature_processing
```

Объект класс сконфигурирован для извлечения информационных образов. Далее для получения образа вызывается метод `get_feature`, которому необходимо передать путь к MP3 или WAV файлу, а также мета-информациию по необходимости. Метод вернёт массив объектов класса `TrackModel`. И для получения вектора информационных образов необходимо вызвать метод `to_vector()`. Если же информационные образы

необходимо сохранить в базе данных MongoDB, то создаётся модуль хранения информационных образов.

Для создания модуля для хранения информациионных образов необходимо создать объект класса `DatabaseModule`, где в конструкторе указывается ip адрес и порт сервера MongoDB.

```
db = DatabaseModule('localhost', 27017)
```

Сохранения информационного образа происходит с помощью вызова метода `store`.

Для использования модуля жанровой классификациии необходимо создать объект класса `GenreClassificationModule`, где в конструкторе указывается следующие параметры:

- `labels_name` – массив строк содержащий название жанров;
- `cv` – количество разбиений при перекрёстной проверки;
- `classifiers` – ассоциативный массив методов классификации, где ключом является название метода, а значением – объект класса;

```
clf = {
   'Nearest Neighbors 3': KNeighborsClassifier(3),
   'Decision Tree': DecisionTreeClassifier(),
   'AdaBoost': AdaBoostClassifier(),
   'Gaussian Naive Bayes': GaussianNB(),
   'QDA': QuadraticDiscriminantAnalysis(),
   ...
}

clf_module = GenreClassificationModule(
              cv=10,
              labels_name=genre_list,
              classifiers=clf)
```

Затем вызвать метод `classify`, который вернёт ассоциативный массив, где ключ – это название метода классификации, а значение массив содержащий МО оценки перекрёстной проверки, СКО этой оценки и матрицу ошибок классификации. Метод `classify` принимает первым значением массив информационных образов, вторым – их кассы и метаинформацию о образе. Метаинформации используется при классификации всего трека, а не отдельных фрагметов. Перед началом выполнения классификации рекомендуется нормализовать данные по каждому признаку.

```
X, Y, meta = load_data()

result = module.classify(X, Y, meta)
```

Для создания модуля визуализации необходимо создать объект класса `VisualizeDataModule`, который имеет два метода: `plot_2d` – для

отображение в двумерную плоскость и `plot_3d` – для отображения пространстав признаков в трёхмерное пространство. И меют одинаковую сигнатуру. Первым аргументом идёт массив информационных образов, вторым – их классы, флаг отобораження в отдельном окне и метод уменьшения размерности. Класс поддерживает два метода уменьшения размерности: t-SNE для нелинейного распределения и PCA (метод главных компонент) для линейного.

```
visualizer = VisualizeDataModule()

visualizer.plot_2d(X, labels=Y,
                genre_list=genre_list,
                reduction_method='PCA')
visualizer.plot_3d(X, labels=Y,
                genre_list=genre_list,
                reduction_method='t-SNE')
```

9 ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ МОДУЛЯ ВЫДЕЛЕНИЯ ИНФОРМАЦИОННЫХ ОБРАЗОВ ИЗ МУЗЫКАЛЬНОГО ПРОИЗВЕДЕНИЯ

9.1 Описание функций, назначения и потенциальных пользователей ПО

В рамках дипломного проекта был разработан модуль выделения информационных признаков из музыкального произведения.

Программный модуль решает задачи выделение спектральных, временных и иных признаков из музыкального трека, проверка значимости признаков путём использовании их в задаче жанровой классификации, визуализация данных алгоритмом t-SNE.

Целью данного дипломного проекта является создание модуля, который выделял информационные образы из музыкального трека только на основании акустического анализа

Разрабатываемый программный продукт относится к категории программного обеспечения, разрабатываемого по индивидуальному заказу для использования внутри организации-заказчика.

Актуальность разработки данного ПО объясняется тем, существует потребность в наличии доступного инструмента, позволяющего выделять информационные образы из музыкального трека только на основании акустического анализа.

Целью экономического обоснования программных средств является определение экономической выгоды создания данного продукта и дальнейшего его применения.

9.2 Расчёт затрат на разработку ПО

Данный модуль может быть использован в сервисе рекомендации, который позволит значительно улучшить качество рекомендаций. Также потенциальным пользователем, может быть любой человек, который хочет анализировать данные в музыкальной предметной области. Разработка программного средства прозводится по индивидуальному заказу организации-заказчика, однако в будущей перспективе может выйти для свободной реализации на рынке.

Для оценки экономической эффективности разработанного программного обеспечения проводится расчет затрат на разработку приложения, оценка прибыли от продажи одного такого приложения и расчет показателей эффективности инвестиций в его разработку.

Разрабатываемый программный продукт относится ко второй категории сложности, так как существует возможность переносимости программного обеспечения.

По степени новизны разрабатываемая система автоматизации относится к категории В с коэффициентом новизны Кн = 0,7, так как было выявлено несколько аналогов. Но стоит принять во внимание тот факт, что данный проект не подразумевает в себе использование принципиально нового типа электронно-вычислительных машин, операционных систем.

При разработке проекта используются существующие технологии и средства разработки, которые охватывают около 20 – 30% реализуемых функций, поэтому коэффициент использования стандартных модулей принимается равным 0,8.

При расчете сметы затрат будут использоваться данные, приведенные в таблице 9.1.

Таблица 9.1 – Исходные данные для расчета

Наименования показателей	Буквенные обозначения	Ед.измерения	Количество
1	2	3	4
Фонд социальной защиты населения (от заработной платы)	$Н_{соц}$	%	34
Обязательное страхование (от несчастных случаев на производстве, от заработной платы)	$Н_{стр}$	%	0,6
Налог на прибыль	$Н_{приб}$	%	18
Налог на недвижимость (от стоимости зданий и сооружений)	$Н_{недв}$	%	1
НДС (Налог на добавленную стоимость)	НДС	%	20
Норма дисконта	$Е_{н}$	%	17
Тарифная ставка 1-го разряда	$Т_{м1}$	руб.	31
Часовая тарифная ставка 1-го разряда	$Т_{ч}$	руб.	0,19
Установленный фонд рабочего времени	$Ф_{рв}$	часов	166
Продолжительность рабочего дня	$Т_{ч}$	часов	8
Тарифный коэффициент	$Т_{к}$	-	2,03
Коэффициент премирования	$К_{п}$	единиц	1,5

Основой для расчёта сметы затрат является основная заработная плата разработчиков проекта. Затраты на основную заработную плату команды разработчиков определяются исходя из состава и численности команды, раз-

меров месячной заработной платы каждого из участников команды, а также общей трудоемкости разработки программного обеспечения.

Для осуществления упрощённого расчёта затрат на разработку ПО следует произвести расчёт следующих статей:

- затраты на основную заработную плату разработчиков;
- затраты на дополнительную заработную плату разработчиков;
- отчисления на социальные нужды;
- прочие затраты (амортизация оборудования, расходы на электроэнергию, командировочные расходы, накладные расходы и т.п.).

Расчёт затрат на основную заработную плату разработчиков осуществляется на основе численности и состава команды, размеров месячной заработной платы каждого из участников команды, а также общей трудоёмкости процесса разработки программного обеспечения.

В данном случае имеются четыре работника – два программиста I-й категории, руководитель проекта и специалист по анализу данных. Месячная тарифная ставка каждого исполнителя определяется по формуле:

$$\mathrm{Т_м} = \mathrm{Т_{м1}} \cdot \mathrm{Т_к}\,, \tag{9.1}$$

где $\mathrm{Т_{м1}}$ — месячная тарифная ставка первого разряда, руб.;
$\mathrm{Т_к}$ — тарифный коэффициент.

Месячная тарифная ставка руководителя проекта составит:

$$\mathrm{Т_{м.рук}} = 19{,}35 \times 31 = 600 \text{ руб.} \tag{9.2}$$

Месячная тарифная ставка программиста составит:

$$\mathrm{Т_{м.прог}} = 16{,}13 \times 31 = 500 \text{ руб.} \tag{9.3}$$

Месячная тарифная ставка специалиста по анализу данных составит:

$$\mathrm{Т_{м.спец}} = 19{,}35 \times 31 = 600 \text{ руб.} \tag{9.4}$$

Исходя из месячной тарифной ставки рассчитывается часовая тарифная ставка:

$$\mathrm{Т_ч} = \frac{\mathrm{Т_м}}{\mathrm{Ф_р}}\,, \tag{9.5}$$

Часовая тарифная ставка руководителя проекта в соответствии с формулой составит:

$$\mathrm{Т_{ч.рук}} = \frac{600}{166} = 3{,}61 \text{ руб.} \tag{9.6}$$

Часовая тарифная ставка программиста первой категории в соответствии с формулой составит:

$$\mathrm{Т_{ч.прог}} = \frac{500}{166} = 3{,}01 \text{ руб.} \tag{9.7}$$

Часовая тарифная ставка специалиста по анализу данных в соответствии с формулой составит:

$$\text{Т}_{\text{ч.спец}} = \frac{600}{166} = 3{,}61 \text{ руб.} \tag{9.8}$$

Основная заработная плата исполнителей на конкретное программное средство является суммой заработных плат каждого из исполнителей в отдельности и определяется по формуле:

$$\text{З}_{\text{о}} = \sum_{i=1}^{n} \text{Т}_{\text{чі}} \cdot \text{t}_{\text{і}}\,, \tag{9.9}$$

где n — количество исполнителей, занятых разработкой конкретного ПО;

$\text{Т}_{\text{чі}}$ — часовая тарифная ставка i-го исполнителя, руб;

$\text{t}_{\text{і}}$ — трудоемкость работ, выполняемых i-го исполнителем, час.

Для руководителя, при заработной плате равной 600 рублей, часовая заработная плата равна 3,61 рубля. Для разработчика, при заработной плате равно 500 рублей, часовая заработная плата равна 3,01 рубля. Для специалиста по анализу данных, часовая заработная плата равна 3,61. Трудоемкость определяется исходя из сложности разработки программного продукта и объема выполняемых им функций. В нашем случае она составляет 90 дней или 720 часов. Тогда основная зарплата исполнителей равна:

$$\text{З}_{\text{о}} = (3{,}01 + 3{,}61 + 3{,}01 + 3{,}61) \times 720 = 9532{,}8 \text{ руб.} \tag{9.10}$$

Затраты на дополнительную заработную плату команды разработчиков включает выплаты, предусмотренные законодательством о труде (оплата отпусков, льготных часов, времени выполнения государственных обязанностей и других выплат, не связанных с основной деятельностью исполнителей), и определяется по формуле:

$$\text{З}_{\text{д}} = \frac{\text{З}_{\text{о}} \cdot \text{Н}_{\text{д}}}{100}\,, \tag{9.11}$$

где $\text{З}_{\text{о}}$ — затраты на основную заработную плату с учетом премии, руб;

$\text{Н}_{\text{д}}$ — норматив дополнительной заработной платы, 15 %.

В результате подстановки получим:

$$\text{З}_{\text{д}} = \frac{14\,299{,}2 \times 15}{100} = 2144{,}88 \text{ руб.} \tag{9.12}$$

Отчисления на социальные нужды включают в предусмотренные законодательством отчисления в фонд социальной защиты (34%) и фонд обязательного страхования (0,6%) в процентах от основной и дополнительной

Таблица 9.2 – Расчет затрат на основную заработную плату команды

№	Участник команды	Выполняемые работы	Месячная заработная плата, р	Часовая заработная плата, р.	Трудоемкость работ, ч.	Основная заработная плата, р.
1	Руководитель проекта	Контроль, помощь	600	3,61	720	2599,2
2	Программист 1-й категории	Разработка	500	3,01	720	2167,2
3	Программист 1-й категории	Разработка	500	3,01	720	2167,2
4	Специалиста по анализу данных	Разработка	600	3,61	720	2599,2
ПРЕМИЯ (50%)						4766,4
Итого затраты на основную заработную плату разработчиков						14 299,2

заработной платы и вычисляются по формуле:

$$З_{соц} = \frac{(З_о + З_д) \cdot Н_{соц}}{100}, \tag{9.13}$$

где $Н_{соц}$ — норматив отчислений на социальные нужды 34% и норматив отчислений в фонд социального страхования, 0,6 % .

$$З_{соц} = \frac{(14\,299{,}2 + 2144{,}88) \times 34{,}6}{100} = 5689{,}65 \text{ руб.} \tag{9.14}$$

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по формуле:

$$З_{пз} = \frac{З_о \cdot Н_{пз}}{100}, \tag{9.15}$$

где $Н_{пз}$ — норматив прочих затрат в целом по организации, 100 % .

$$З_{пз} = \frac{14\,299{,}2 \times 100}{100} = 14\,299{,}2 \text{ руб.} \tag{9.16}$$

Общая сумма расходов по смете определяется:

$$З_п = З_о + З_д + З_{соц} + Р_{пр}, \tag{9.17}$$

Подставив рассчитанные ранее значения в формулу, получим:

$$З_{п} = 14\,299{,}2 + 2144{,}88 + 5689{,}65 + 14\,299{,}2 = 36\,432{,}93 \text{ руб.} \qquad (9.18)$$

Полная сумма затрат на разработку программного обеспечения находится путем суммирования всех рассчитанных статей затрат (таблица 9.3).

Таблица 9.3 – Затраты на разработку программного обеспечения

Статья затрат	Сумма, руб.
Основная заработная плата команды разработчиков	14 299,2
Дополнительная заработная плата команды разработчиков	2144,88
Отчисления на социальные нужды	5689,65
Прочие затраты	14 299,2
Общая сумма затрат на разработку	36 432,93

9.3 Оценка результата (эффекта) от продажи ПО

Экономический эффект представляет собой прирост чистой прибыли, полученный организацией в результате использования разработанного ПО. Как правило, он может быть достигнут за счет:

- уменьшения (экономии) затрат на заработную плату за счет замены «ручных» операций и бизнес-процессов информационной системой;
- ускорения скорости обслуживания клиентов и рост возможности обслуживания большего их количества в единицу времени, т.е. рост производительности труда;
- появления нового канала сбыта продукции или получения заказов (как в случае внедрения интернет-магазина);
- и т.п.

Экономический эффект организации-разработчика программного обеспечения в данном случае заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного ПО.

При свободной реализации на рынке IT экономический эффект заключается в получении прибыли от его продажи множеству потребителей. Прибыль от реализации в данном случае напрямую зависит от объемов продаж, цены реализации и затрат на разработку данного проекта. Организация, осуществляющая финансово-хозяйственную деятельность, заинтересована не только в наибольшей величине прибыли, но и в отдаче вложенных средств. Отдача или эффективность вложенных средств характеризуется размером

прибыли, получаемой предприятием. Показатели рентабельности характеризуют эффективность работы организации в целом, а также доходность различных направлений деятельности.

Далее следует определить цену на одну копию (лицензию) ПО. Цена формируется на основе затрат на разработку и реализацию ПО. Тогда расчет прибыли от продажи одной копии (лицензии) ПО осуществляется по формуле:

$$Ц = П_{ед} + НДС + \frac{З_р}{N}, \tag{9.19}$$

где Ц — цена реализации одной копии (лицензии) ПО, руб.;
$З_р$ — сумма расходов на разработку и реализацию программного обеспечения, руб.В дипломном проекте расходы на реализацию приняты равным 5%;
N — количество копий (лицензий) ПО, которое будет куплено клиентами за год, руб.;
$П_{ед}$ — прибыль, получаемая организацией-разработчиком от реализации одной копии программного продукта, руб.;
НДС — сумма налога на добавленную стоимость, руб.;

Прибыль от продажи одной копии (лицензии) ПО осуществляется по формуле:

$$П_{ед} = \frac{С_п \cdot У_р}{100 \times N}, \tag{9.20}$$

где $С_п$ — себестоимость ПО, руб.;
$У_р$ — запланированный уровень рентабельности, 35 %.

$$П_{ед} = \frac{36\,432{,}93 \times 35}{100 \times 20} = 637{,}6 \text{ руб.} \tag{9.21}$$

Цена одной копии (лицензии) программного обеспечения данного направления составляет 637,6 руб.

Следовательно годовая прибыль составит:

$$П = 637{,}6 \times 20 = 12\,751{,}5 \text{ руб.} \tag{9.22}$$

Подставив рассчитанные ранее значения в формулу, получим:

$$Ц = 637{,}6 + \frac{36\,432{,}93 \times 1{,}05}{20} = 2550{,}32 \text{ руб.} \tag{9.23}$$

Рентабельность затрат рассчитаем по формуле. Проект будет экономически эффективным, если рентабельность затрат на разработку программного средства будет не меньше средней процентной ставки по банковским депозитным вкладам.

$$Р = 100 \cdot \frac{П}{З_р}, \tag{9.24}$$

где П — годовая прибыль, руб.;
$З_р$ — сумма расходов на разработку и реализацию, руб.

$$Р = 100 \times \frac{12\,751{,}5}{36\,432{,}93} = 35\%. \tag{9.25}$$

Средняя процентная ставка по депозиту за март 2017 года составляет 7,9% для юридических лиц, что меньше рентабельности данного проекта. Соответственно проект является экономически эффективным. Учитывая налог на прибыль, можно рассчитать итоговую сумму, которая останется разработчику и будет является его экономическим эффектом:

$$ЧП = П - \frac{П \cdot Н_{приб}}{100}, \tag{9.26}$$

где $Н_{приб}$ — ставка налога на прибыль, 18 % ;
П — прогнозируемая прибыль, руб..

$$ЧП = 12\,751{,}5 - \frac{12\,751{,}5 \cdot 18}{100} = 10\,456{,}23 \text{ руб.} \tag{9.27}$$

Чистая прибыль от реализации ПО (ЧП = 10456,23 рублей) остается организации-разработчику и представляет собой экономический эффект.

9.4 Расчёт показателей эффективности инвестиций в разработку ПО

Так как сумма инвестиций в разработку ПО больше суммы годового экономического эффекта, то экономическая целесообразность инвестиций в разработку и использование программного средства осуществляется на основе расчёта и оценки следующих показателей:

- чистый дисконтированный доход (ЧДД);
- срок окупаемости инвестиций;
- рентабельность инвестиций.

Метод чистой дисконтированной доходности основан на сопоставлении дисконтированной стоимости денежных поступлений (инвестиций), генерируемых предприятием в течение прогнозируемого периода. Целью данного метода является выявление реального размера прибыли, который может быть получен организацией вследствие реализации данного инвестиционного проекта.

Коэффициент дисконтирования соответствующего года t определяется по формуле:

$$a_t = \frac{1}{(1 + E_n)^t}, \tag{9.28}$$

где E_n — норма дисконта, равная или больше средней процентной ставки по банковским депозитам, действующей на момент проведения расчётов;

t — порядковый номер года периода реализации инвестиционного проекта (1 – 2017, 2 – 2018, 3 – 2019, 4 – 2020).

Подставляя значения, получим значения коэффициентов дисконтирования для 2017 – 2020 годов:

$$a_1 = \frac{1}{(1+0{,}17)^1} = 0{,}85\,, \tag{9.29}$$

$$a_2 = \frac{1}{(1+0{,}17)^2} = 0{,}73\,, \tag{9.30}$$

$$a_3 = \frac{1}{(1+0{,}17)^3} = 0{,}62\,, \tag{9.31}$$

$$a_4 = \frac{1}{(1+0{,}17)^4} = 0{,}53\,. \tag{9.32}$$

Чистый дисконтированный доход рассчитывается по следующей формуле:

$$\text{ЧДД} = \sum_{t=1}^{n} (P_t \cdot a_t - З_t \cdot a_t)\,, \tag{9.33}$$

где n — расчётный период, лет;

P_t — результат (экономический эффект), полученный в году t, руб.;

$З_t$ — затраты (инвестиции в разработку ПО) в году t, руб.

$$\text{ЧДД} = -20\,129{,}21 + 9308{,}59 + 7905{,}93 + 6758{,}29 = 3843{,}6 \text{ руб.} \tag{9.34}$$

Рентабельность инвестиций рассчитывается по формуле:

$$P_и = \frac{\sum_{t=0}^{n} P_t \cdot a_t}{\sum_{t=0}^{n} З_t \cdot a_t} \cdot 100\%\,, \tag{9.35}$$

По формуле рентабельность инвестиций равна:

$$P_и = \frac{10\,838{,}77 + 9308{,}59 + 7905{,}93 + 6758{,}29}{30\,967{,}99} \cdot 100 = 112{,}3\%. \tag{9.36}$$

На основании проведенной анализа экономической эффективности разработки системы можно сделать следующие выводы. Из оценки затрат на создание и поддержание программного продукта и прибыли, полученной за продажу копии программы, мы подсчитали рентабельность затрат. Она оказалась выше средней процентной ставки по депозиту, что показывает выгоду от реализации данной продукции. Показатели эффективности

Таблица 9.4 – Расчет эффективности инвестиционного проекта по разработке программного обеспечения

Показатели	2017	2018	2019	2020
РЕЗУЛЬТАТ				
Экономический эффект	12 751,5	12 751,5	12 751,5	12 751,5
Дисконтированный результат	10 838,77	9308,59	7905,93	6758,29
ЗАТРАТЫ				
Инвестиции в разработку программного средства	36 432,93	0	0	0
Дисконтированные инвестиции	30 967,99	0	0	0
Чистый дисконтированный доход по годам	−20 129,21	9308,59	7905,93	6758,29
Дисконтированный результат	−20 129,21	−10 819,6	−2913,69	3843,6
Коэффициент дисконтирования	0,85	0,73	0,62	0,53

инвестиций на создание программного обеспечения в результате подсчетов показали, что разработка данного программного продукта является экономически целесообразной.

ЗАКЛЮЧЕНИЕ

В результате дипломного проектирования были проведены исследования в области выделения информационных образов и сервисов рекомендации. Было проведено исследование существующих способов выделения информационных признаков при решении задач: распознования неперкуссионных инструментов, распознования заимствований и классификации жанров. Были рассмотрены существующие сервисы и приложения рекомендации музыки на основе акустического анализа, рассмотрены их положительные и отрицательные стороны. На основе исследования были выбраны временные, спектральные, ритмические(перкуссионные) признаки, а также мел-кепстаральные коэффициенты.

Был спроектирован и реализован программный модуль по выделению выбранных признаков. Выделенные признаки были использованы для решения задачи жанровой классификации для 10 жанров музыки с помощью различных методов классификации. Было проведено сравнение эффективности методов классификации как отдельных фрагментов, так и целого трека. Решение задачи жанровой классификации показало, что выбранные признаки значимы и на их основе можно делать рекомендацию. Также на были выделены жанры, которые лучше всего выделяются всеми методами классификации. Для этих жанров была сделана визуализации в двумерном и трёхмерном измерении путйм уменьшение размерености векторов признаков алгоритмами t-SNE и PCA.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Yandex N. V. Десять миллионов треков на Яндекс.Музыке [Электронный ресурс]. — Электронные данные. — Режим доступа: https://yandex.ru/blog/company/69072. — Дата доступа: 05.02.2017.

2 Kevin Murnane. The US Music Industry Crossed A Threshold In 2016 [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.forbes.com/sites/kevinmurnane/2017/01/18/the-us-music-industry-passed-a-milestone-in-2016/#312060b15a90. — Дата доступа: 05.02.2017.

3 Yandex N. V. Как это работает? Рекомендации в Яндекс.Музыке [Электронный ресурс]. — Электронные данные. — Режим доступа: https://yandex.ru/blog/company/92883. — Дата доступа: 05.02.2017.

4 M. Jones. Introduction to approaches and algorithms [Электронный ресурс]. — Электронные данные. — Режим доступа: https://www.ibm.com/developerworks/opensource/library/os-recommender1/index.html?S_TACT=105AGX99&S_CMP=CP. — Дата доступа: 05.02.2017.

5 George Tzanetakis Georg Essl, Perry Cook. Automatic Musical Genre Classification Of Audio Signals. — 2001.

6 Bashi, Jamil George. Music Similarity Measures for Interpolative Playlist Generation. — 2008.

7 Balen, Jan Van. Automatic Recognition of Samples in Musical Audio. — 2011.

8 Martin, Keith Dana. Sound-Source Recognition: A Theory and Computational Model. — 1999.

9 Brown, Judith C. Calculation of a constant Q spectral transform. — 1991.

10 Ian Glover, Peter Grant. Digital Communications. — 1998.

11 Mohit Rajani, Luke Ekkizogloy. Supervised Learning in Genre Classification. — 2009.

12 Logan, Beth. Mel Frequency Cepstral Coefficient for Music Modeling. — 2000.

13 HOLO — Система анализа музыки [Электронный ресурс]. — Электронные данные. — Режим доступа: https://habrahabr.ru/post/194724/. — Дата доступа: 05.02.2017.

14 Формирование музыкальных предпочтений у нейронной сети — эксперимент по созданию умного плеера [Электронный ресурс]. — Электонные данные. — Режим доступа: https://habrahabr.ru/post/263811/. — Дата доступа: 05.02.2017.

15 About Pandora [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.pandora.com/about. — Дата доступа: 05.02.2017.

16 The Best Music Services Compared [Электронный ресурс]. — Электронные данные. — Режим доступа:

http://www.techlicious.com/guide/best-music-service-best-for-you/. — Дата доступа: 05.02.2017.

17 Consumer item matching method and system US 7003515 B1 [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.google.com/patents/US7003515?dq=7,003,515. — Дата доступа: 05.02.2017.

18 Tim Westergren (Music Genome Project Founder) [Электронный ресурс]. — Электронные данные. — Режим доступа: http://www.tinymixtapes.com/features/tim-westergren-music-genome-project-founder. — Дата доступа: 05.02.2017.

ПРИЛОЖЕНИЕ А
(*обязательное)*

Листинг класса PreprocessingModule

ПРИЛОЖЕНИЕ Б
(*обязательное)*

Листинг класса SpectralTransformer

ПРИЛОЖЕНИЕ В
(*обязательное)*

Спецификация

ПРИЛОЖЕНИЕ Г
(*обязательное)*

Ведомость документов

```
class PreprocessingModule:
   alpha = 0.0
   overlap = 0.0
   cut_start = 0.0
   cut_end = 0.0
   frame_size_sec = 0

   def __init__(self, alpha, overlap, cut_start, cut_end,
      frame_size_sec):
      self.alpha = alpha
      self.overlap = overlap
      self.cut_end = cut_end
      self.cut_start = cut_start
      self.frame_size_sec = frame_size_sec

   def scale(self, track):
      track.data = track.data.astype('float64')
      track.data = scale(track.data, with_std=True, with_mean=
         True)
      return track

   def stereo_to_mono(self, track):
      if len(track.data.shape) > 1:
         track.data = np.mean(track.data, axis=0)
      return track

   def filter(self, track):
      fltr = LowPassSinglePole(self.alpha)
      filter = np.vectorize(lambda x: fltr.filter(x))
      track.data = filter(track.data)
      return track

   def framing(self, track):
      frame_size = self.frame_size_sec * track.sample_rate
      data = track.data
      results = []
      iteration = int((1 - self.overlap) * frame_size)
      stop = (int(len(data) / iteration) - 1) * iteration
      for i in range(0, stop, iteration):
         results.append(Track((track.sample_rate, data[i:i +
            frame_size]), track.label))
      return results

   def cutting(self, track):
      length = len(track.data)
      track.data = track.data[int(length * self.cut_start)
      : int(length * (1 - self.cut_end))]
      return track
```

```
class SpectralTransformer:
   window = signal.hamming(1024)
   level = 4
   alpha = 0.99
   rate = 16

   def __init__(self, window, level, alpha, rate):
      self.window = window
      self.level = level
      self.alpha = alpha
      self.rate = rate

   def short_time_fourier(self, track):

      f, t, Zxx = signal.stft(track.data,
                        window=self.window,
                        nperseg=len(self.window))
      return np.abs(Zxx)


   def wavelet_daubechies(self, data):
      data = np.array(pywt.swt(data, 'db4', level=self.level))
      data = np.array([np.sqrt(np.power(i[0], 2) +
                         np.power(i[1], 2)) for i in data])
      data = data.reshape(self.level, data.shape[-1])
      return data

   def __round_to_power_of_two(self, data):
      size = len(data)
      new_size = 2 ** (size.bit_length() - 1)
      return data[:new_size]

   def filter(self, data):
      fltr = LowPassSinglePole(self.alpha)
      result = []
      for i in data:
         result.append(fltr.filter(i))
      return np.array(result)

   def resampling(self, data):
      return data[::self.rate]

   def normalize_and_sum(self, data):
      data = np.array(data)
      accumulator = np.zeros(data.shape[1])
      for i in data:
         accumulator += scale(i, with_mean=True, with_std=False)
      return accumulator

   def autocorrelation(self, data):
      data = scipy.fft(data)
```

```
        data = np.abs(scipy.ifft(data * data)) / len(data) / self.
            level
        return data

    def percussion_correlogramm(self, track):
        data = self.__round_to_power_of_two(track.data)
        data = self.wavelet_daubechies(data)
        results = []
        for i in data:
            filtered = self.filter(i)
            resampled = self.resampling(filtered)
            results.append(resampled)
        data = self.normalize_and_sum(results)
        return data
```