

Министерство образования Республики Беларусь

Учреждение образования  
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей  
Кафедра информатики

*К защите допустить:*

Заведующая кафедрой  
информатики

\_\_\_\_\_ Н. А. Волорова

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

к дипломному проекту  
на тему:

**СЕРВИС ДЛЯ ИЗВЛЕЧЕНИЯ ХАРАКТЕРИСТИК  
МУЗЫКАЛЬНОЙ КОМПОЗИЦИИ С ПРИМЕНЕНИЕМ  
МАШИННОГО ОБУЧЕНИЯ**

БГУИР ДП 1-40 04 01 097 ПЗ

Студент

Д. В. Шматков

Руководитель

П. С. Саттарова

Консультанты:

*от кафедры Информатики*

П. С. Саттарова

*по экономической части*

К. Р. Литвинович

Нормоконтролёр

Н. Н. Бабенко

Рецензент

Минск 2017

## РЕФЕРАТ

Пояснительная записка 46 с., 8 рис., 5 табл., 30 формул, 6 источников.  
ПРОГРАММНОЕ СРЕДСТВО, ВЕБ-СЕРВИС, УНИВЕРСИТЕТ, АНАЛИЗ  
МУЗЫКАЛЬНЫХ КОМПОЗИЦИЙ, МАШИННОЕ ОБУЧЕНИЕ,  
ИНДИВИДУАЛЬНЫЕ ЗАДАНИЯ

Цель настоящего дипломного проекта состоит в разработке программной системы, предназначенной для автоматизации извлечения характеристик музыкальных произведений.

В процессе анализа предметной области были выделены основные характеристики музыкальных композиций и способы их извлечения, которые в настоящее время практически не охвачены автоматизацией. Было проведено их исследование и моделирование. Кроме того, рассмотрены существующие средства, разрозненно разрабатываемые и применяемые сотрудниками университетов и отдельными компаниями (так называемые частичные аналоги). Выработаны функциональные и нефункциональные требования.

Была разработана архитектура программной системы, для каждой ее составной части было проведено разграничение реализуемых задач, проектирование, уточнение используемых технологий и собственно разработка. Были выбраны наиболее современные средства разработки, широко применяемые в индустрии.

Полученные в ходе технико-экономического обоснования результаты о прибыли для разработчика, пользователя, уровень рентабельности, а также экономический эффект доказывают целесообразность разработки проекта.

## СОДЕРЖАНИЕ

Введение . . . . .	8
1 Анализ предметной области . . . . .	9
2 Обзор существующих аналогов . . . . .	13
2.1 The Echo Nest . . . . .	13
2.2 Niland . . . . .	13
2.3 Music Technology Group . . . . .	14
2.4 DeepMind . . . . .	15
3 Проектирование программного средства . . . . .	16
3.1 Требования к проектируемому программному средству . . . . .	16
3.2 Построение архитектуры . . . . .	17
4 Используемые технологии . . . . .	22
4.1 Essentia . . . . .	22
4.2 Python . . . . .	22
4.3 TensorFlow . . . . .	23
4.4 Keras . . . . .	24
4.5 Flask . . . . .	25
4.6 Gunicorn . . . . .	26
4.7 Supervisor . . . . .	26
4.8 Docker . . . . .	26
5 Тренировка моделей машинного обучения . . . . .	28
6 Руководство по установке и использованию . . . . .	30
6.1 Установка . . . . .	30
6.2 Использование . . . . .	31
7 Техничко-экономическое обоснование разработки и внедрения программного средства . . . . .	32
7.1 Характеристика программного средства . . . . .	32
7.2 Определение объема и трудоемкости ПС . . . . .	32
7.3 Расчет сметы затрат . . . . .	35
7.4 Оценка экономической эффективности применения ПС у пользователя . . . . .	39
Заключение . . . . .	43
Список использованных источников . . . . .	44
Приложение А Фрагменты исходного кода . . . . .	45

## ОПРЕДЕЛЕНИЯ И СОКРАЩЕНИЯ

В настоящей пояснительной записке применяются следующие определения и сокращения.

*API* – Application Programming Interface (программный интерфейс приложения).

*t-SNE* – алгоритм машинного обучения, который относят к методам множественного обучения признаков [1].

*music2vec* – английское сокращение от music to vector (композиция в вектор).

*MP3* – кодек третьего уровня, разработанный командой MPEG, формат файла для хранения аудиоинформации.

*WAV* – формат файла-контейнера для хранения записи оцифрованного аудиопотока.

*FLAC* – популярный свободный кодек, предназначенный для сжатия аудиоданных без потерь.

*M4A* – проприетарный (патентованный) формат аудиофайла с потерями.

*JSON* – JavaScript Object Notation (текстовый формат обмена данными, основанный на JavaScript).

*XML* – eXtensible Markup Language (расширяемый язык разметки).

*JVM* – Java Virtual Machine (основная часть исполняющей системы Java).

*MSIL* – Microsoft Intermediate Language (внутренний системный язык программирования Microsoft).

*LLVM* – Low Level Virtual Machine (универсальная система анализа, трансформации и оптимизации программ).

*JIT* – динамическая компиляция, технология увеличения производительности программных систем, использующих байт-код, путём компиляции байт-кода в машинный код или в другой формат непосредственно во время работы программы.

*PyPy* – интерпретатор языка программирования Python.

*ANSI* – американский национальный институт стандартов.

*ISO* – международная организация по стандартизации.

*TPU* – специализированные интегральные схемы, разработанные специально для машинного обучения.

*ASIC* – интегральная схема, специализированная для решения конкретной задачи.

*ИИ* – искусственный интеллект.

*cgroups* – механизм ядра Linux, который ограничивает и изолирует вычислительные ресурсы (процессорные, сетевые, ресурсы памяти, ресурсы ввода-вывода) для групп процессов.

*LXC* – система виртуализации на уровне операционной системы для запуска нескольких изолированных экземпляров операционной системы Linux на одном узле.

*HTTP* – HyperText Transfer Protocol (протокол прикладного уровня передачи данных).

*WSGI* – Web Server Gateway Interface (стандарт взаимодействия между Python-программой, выполняющейся на стороне сервера, и самим веб-сервером).

*UNIX* – семейство переносимых, многозадачных и многопользовательских операционных систем.

*fork* – системный вызов, создающий новый процесс (потомок), который является практически полной копией процесса-родителя, выполняющего этот вызов.

*exec* – системный вызов, загружающий в пространство процесса новую программу.

*Affero GPLv3* – свободная лицензия, созданная специально для таких программ, как вебприложения, так что пользователи, использующие изменённую программу через сеть, могут получить её исходный код.

*Фреймворк* – программная платформа, определяющая структуру программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта.

*RNN* – рекуррентная нейронная сеть.

*Linux* – семейство Unix-подобных операционных систем на базе ядра Linux, включающих тот или иной набор утилит и программ проекта GNU, и, возможно, другие компоненты.

*Debian* – операционная система, основанная на Linux, состоящая из свободного ПО с открытым исходным кодом.

## ВВЕДЕНИЕ

Подъем популярности сервисов электронного распространения медиainформации дал беспрецедентный доступ пользователям к записям музыкальных композиций. Сейчас такие сервисы, как iTunes, Google Music, Яндекс Музыка, Spotify и другие, предоставляют мгновенный доступ к миллионам записей. Перед пользователями возникает проблема выбора следующей композиции для прослушивания. Для решения этой проблемы, а так же для того, чтобы облегчить ориентацию в этом большом количестве информации, сервисы электронного распространения записей предоставляют пользователям системы рекомендации контента.

Современные системы рекомендации контента обычно используют статистику прослушивания композиций пользователями, чтобы сделать рекомендации более точными. Однако на такие системы часто оказывают влияние смещения популярности в больших масштабах, из-за чего системы не могут рекомендовать композиции, которые менее популярны. Так же это затрудняет создание индивидуальных рекомендаций для пользователей, исходя из их предпочтений. Поэтому возникает необходимость извлекать информацию из композиций, а так же классифицировать сами композиции. На сегодняшний день извлечение характеристик и классификация музыкальных композиций является интересной и соревновательной задачей, которой занимаются различные компании и исследователи. На данный момент сильное развитие получают способы, основанные на машинном обучении.

Целью данного дипломного проекта является создание сервис для извлечения музыкальных характеристик, который будет пригоден для дальнейшей интеграцией с другими системами. В ходе работы предстоит выполнить следующие задачи:

- а) выбрать способ извлечения базовых характеристик музыкальных композиций (таких как спектр, ритм, продолжительность);
- б) построить модель машинного обучения для извлечения высокоуровневых характеристик;
- в) собрать и упорядочить данные для обучения модели;
- г) построить эффективную архитектуру сервиса;
- д) построить API для интеграции с другими системами.

Реализация сервиса для извлечения характеристик музыкальных композиций позволит упростить задачу классификации контента для сервисов электронного распространения записей, а так же создать систему для создания более точных и индивидуальных рекомендаций для пользователей.

## 1 АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

Извлечение музыкальной информации - небольшая, но интересная и быстрорастущая область исследований, которая охватывает большой диапазон продуктов, используемых по всему миру. Эта область охватывает несколько дисциплин научных знаний: музыковедение, психологию, обработку сигналов, машинное обучение и комбинации этих дисциплин. Задача, которую будет решать разрабатываемый сервис, относится к задачам извлечения музыкальной информации.

Несмотря на то, что извлечение музыкальной информации является еще небольшой областью исследований, для извлечения характеристик музыкальных приложений применяется достаточное количество методов. Эти методы различаются как на основании применяемых подходов, так и на основании уровней извлекаемых характеристик. На основании характеристик выделяют методы, которые извлекают низкоуровневые характеристики (спектр, ритм), и методы которые извлекают высокоуровневые характеристики (жанр, настроение). В основном, нас будут интересовать методы, которые извлекают высокоуровневые характеристики.

Для извлечения высокоуровневых характеристик выделяют два класса методов: основанные на математических функциях, в основе которых лежит машинное обучение. В музыке появляются новые направления, а старые могут изменяться, поэтому наиболее перспективными являются методы, которые способны подстраиваться под переменчивую природу музыки. Под такие условия в большей степени подходят методы, в основе которых лежит машинное обучение.

Машинное обучение - класс методов, характерной чертой которых является не прямое решение задачи, а решение, которое строится в процессе обучения при применении решения к множеству сходных задач. В последнее время этот класс методов получил активное развитие и поддержку со стороны крупных компаний и университетов.

Цифровой аудиосигнал можно представить в виде изображения звуковой волны.

Цифровой аудиосигнал представляет собой множество точек, которые расставлены на одинаковом расстоянии друг от друга. Расстояние между точками называется частотой дискретизации. Чем меньше интервал (выше частота дискретизации), тем шире частотный диапазон, который можно закодировать таким образом.

Амплитуда колебаний зависит от времени звука и коррелирует с громкостью звука. А частота колебаний напрямую связана с высотой звука. Для

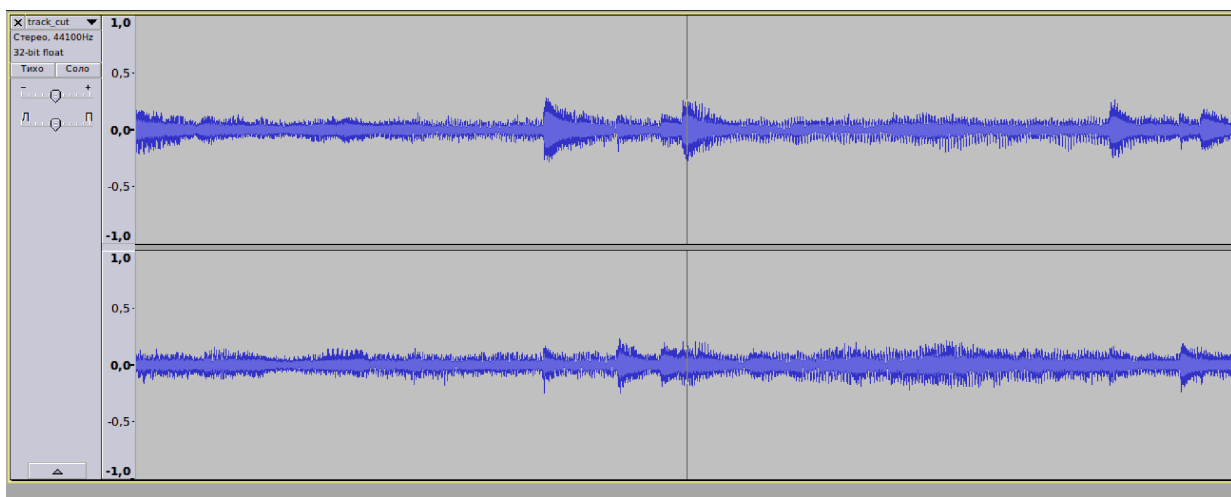


Рисунок 1.1 – Изображение звуковой волны

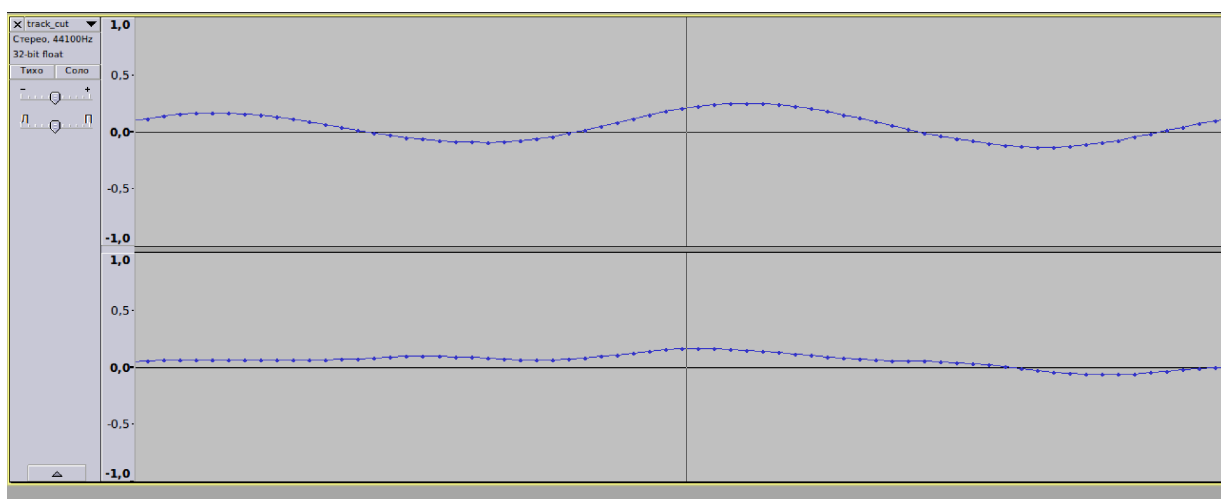


Рисунок 1.2 – Изображение звуковой волны в большом разрешении

того, чтобы получить информацию о частоте колебаний, необходимо применить преобразование Фурье. Преобразование Фурье позволяет разложить периодическую функцию в сумму гармонических с разными частотами. Коэффициенты гармонических функций при сложении будут давать нам те частоты, которые мы хотим получить.

При применении преобразования Фурье ко всей звуковой дорожке мы получим "смазанный" во времени спектр. Для того, чтобы получить спектр, не теряя временной составляющей, необходимо применить к сигналу оконное преобразование Фурье. Оконное преобразование Фурье отличается от обычного тем, что мы делим наш сигнал на короткие отрезки (окна) и применяем к каждому преобразование Фурье. По-сути мы получаем набор спектров - отдельно для каждого отрезка. В результате мы получим картинку, которой можно описать звуковую дорожку.

Жанр композиции является высокоуровневой характеристикой. Для



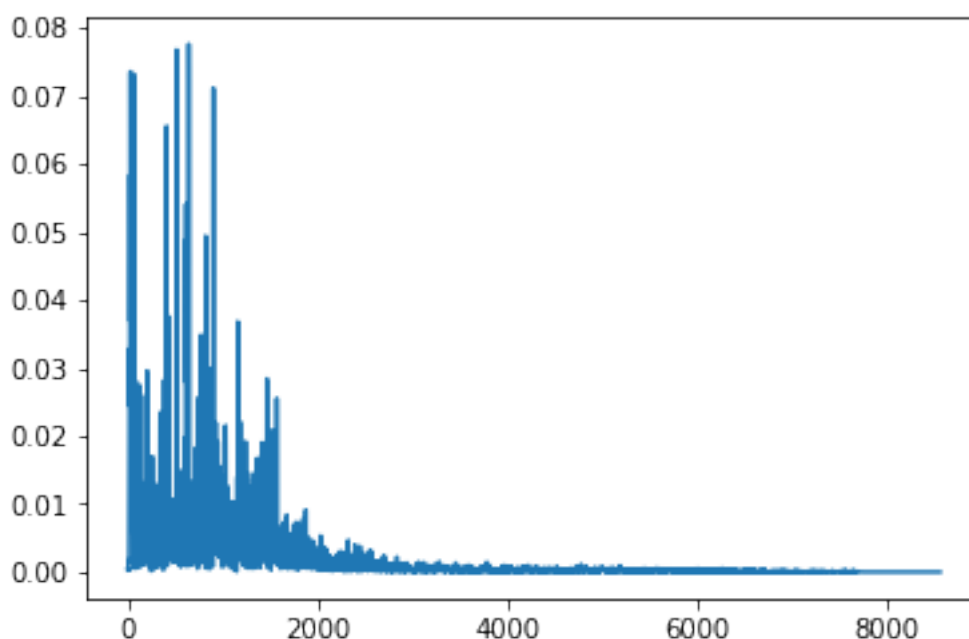


Рисунок 1.3 – Спектральная характеристика

определения жанра необходимо разобраться, как человек воспринимает звук.

Ухо человека устроено так: звуковые волны смещают барабанную перепонку при взаимодействии с ней. Вибрации передаются во внутреннее ухо и считываются им. Смещение барабанной перепонки зависит от звукового давления, при том зависимость является не линейной, а логарифмической. Для измерения громкости принято использовать относительную шкалу - уровень звукового давления (измеряется в децибелах). Так же воспринимаемая громкость зависит и от частоты звука. Для оценки громкости звука используется логарифмическая единица измерения - фон. В шкале фонов, в отличие от децибелов, значения громкости связаны с чувствительностью на разных частотах человеческого уха. Частота 1000 Гц является чистым тоном, и уровень фона для неё численно равен уровню в децибелах. Для остальных частот используют поправки, которые представляют собой стандартизированное семейство кривых, называемых изофонами.

Аналогично с громкостью, частота так же воспринимается человеческим ухом нелинейно. Для измерения воспринимаемой частоты человеческим ухом используется мел-шкала. Шкала основана на статистической обработке субъективного восприятия звука на больших данных. За 1000 мел взят звук с частотой 1000 Гц при уровне громкости 40 фон. За 0 мел взят звук частотой 20 Гц при уровне громкости 40 фон.

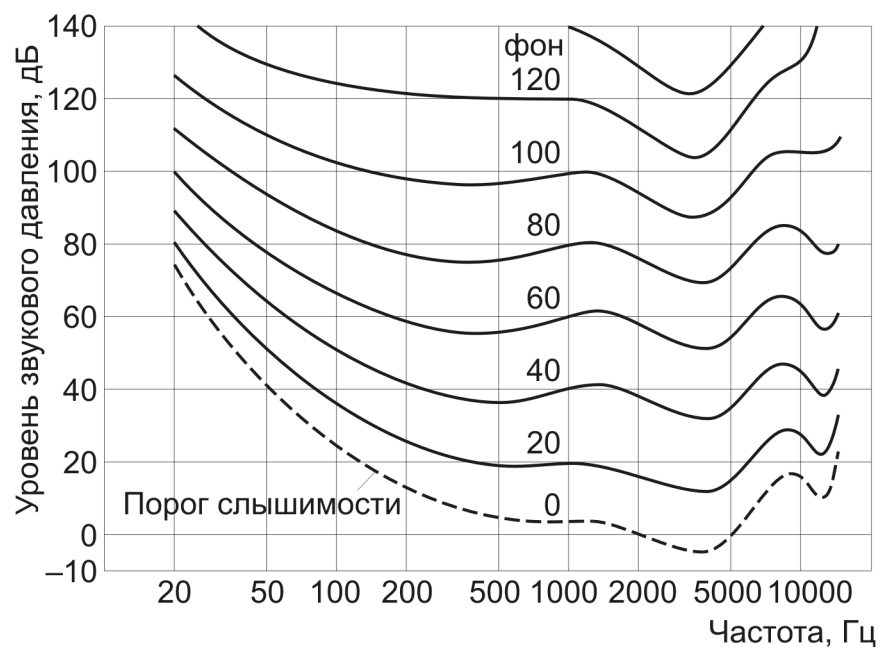


Рисунок 1.4 – Контур равных громкостей

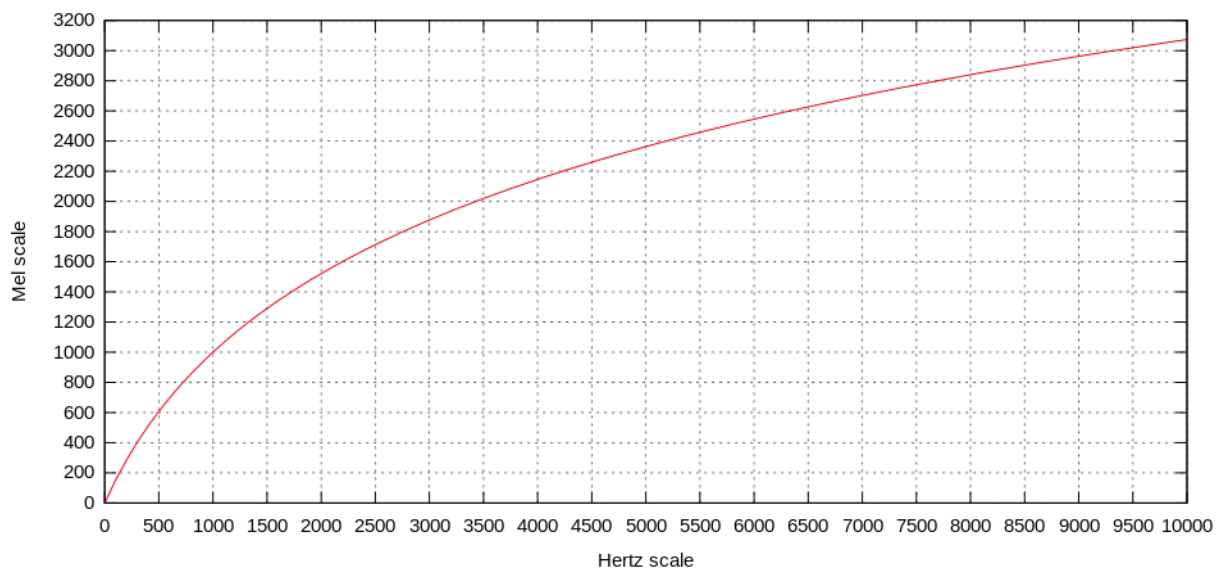


Рисунок 1.5 – Мел шкала

## 2 ОБЗОР СУЩЕСТВУЮЩИХ АНАЛОГОВ

Существует ряд компаний и сервисов, которые решают задачу извлечения характеристик музыкальных композиций и задачи, достаточно близкие к данной. Рассмотрим наиболее популярные и известные из них.

### 2.1 The Echo Nest

The Echo Nest - компания из Соммервила, которая занимается разработкой сервиса для анализа музыкальных композиций и составлении рекомендаций. Их продукт собирает информацию, используя акустический и текстовый анализ. Текстовый анализ состоит в том, что любое упоминание музыкальной композиции, которое найдено в интернете, проходит через системы The Echo Nest. Эти системы настроены на то, чтобы извлекать ключевые слова и термины. Этим данным потом присваивается собственный вес, который говорит об их важности. Акустический анализ начинается с того, что композиция разбивается на небольшие кусочки. Затем для каждого сегмента определяются громкость, тембр и другие характеристики. Далее полученные данные объединяются и анализируются. Анализ проводится с помощью методов машинного обучения, что позволяет понять композицию на высоком уровне. Объединение акустического и текстового анализа позволило создать мощную технологию, которая сделала компанию мировым лидером в алгоритмах анализа музыки. В 2014 году компанию купил мировой гигант стримминга музыки - Spotify.

### 2.2 Niland

Niland - компания из Парижа, которая занимается поиском и рекомендациями музыкальных композиций. Их продукт анализирует композицию, используя акустический анализ. Решаемые задачи разделяются на 2 типа: оценка похожести и классификация. Для оценки похожести используется подход music2vec. То есть преобразование композиции в вектор из значений характеристик. Для классификации композиций используется преобразование музыки в вектор для дальнейшей обработки с помощью алгоритмов машинного обучения. Преобразование музыки в вектор включает в себя:

- а) вычисление спектрограммы, т.е. вычисление интенсивности различных частот в различные моменты времени;
- б) извлечение кратковременных особенностей, т.е. свойств сигнала, которые имеют высокую частоту обновления;

в) моделирование статического распределения кратковременных особенностей и объединение их в вектор.

В результате получается вектор большой размерности, порядка тысячи элементов, который впоследствии используется для классификации композиций и извлечения характеристик высокого уровня.

### 2.3 Music Technology Group

Music Technology Group - исследовательская группа с факультета информации и коммуникационных технологий университета Pompeu Fabra, Барселона. Специализируется на вычислительных задачах в области музыки и звука. Свои исследования группа основывает на знаниях из других областей, таких, как обработка сигналов, машинное обучение и взаимодействие человека с компьютером. Темы исследований, которыми группа занимается в данный момент:

- а) обработка аудиосигналов, т.е. спектральное моделирование для синтеза и трансформации звука;
- б) описание звука и музыки, т.е. семантический анализ и классификация аудио;
- в) продвинутое взаимодействие с музыкой, т.е. разработка интерфейсов для создания и изучения музыки;
- г) звуковые и музыкальные сообщества, т.е. технологии социальных сетей для музыкальных и звуковых приложений.

Группа создала библиотеку для определения степени похожести и классификации музыки, для описания музыки с помощью высокоуровневых характеристик - Gaia. Обработываются данные в несколько этапов: подготовка данных, обучение на подготовленных данных, проверка результатов. Подготовка данных включает в себя:

- а) удаление метаданных;
- б) извлечение тональных характеристик;
- в) нормализацию выделенных характеристик;
- г) преобразование низкоуровневых характеристик в нормально распределенные величины.

Для обучения используется метод опорных векторов с полиномиальным ядром (однородным). В качестве метрики точности используется n-кратная кроссвалидация.

## 2.4 DeepMind

DeepMind - компания, которая занимается искусственным интеллектом. Ранее она была известна под названием Google DeepMind. Компания занимается решением проблем, связанных с интеллектом, т.е. занимается разработкой обучающихся алгоритмов общего назначения. На данный момент компания сосредоточила свои усилия на разработке интеллекта, который способен играть в компьютерные игры - от стратегических до аркад.

Сандер Дилеман, ученый из этой компании, стал соавтором статьи[2], в которой утверждается, что глубинное обучение[3] способно гораздо лучше справляться с рекомендациями, чем колаборативная фильтрация. В качестве инструмента использовалась сверточная нейронная сеть с 7-8 слоями. Для визуализации данных использовался алгоритм t-SNE. Алгоритм определял инструменты, аккорды, и даже гармонии и прогрессии.

### 3 ПРОЕКТИРОВАНИЕ ПРОГРАММНОГО СРЕДСТВА

#### 3.1 Требования к проектируемому программному средству

По результатам изучения предметной области, анализа литературных источников и обзора существующих систем-аналогов сформулируем требования к проектируемому программному средству.

Сервис должен предоставлять простой API для извлечения характеристик из аудиозаписей. Для поддержки большего числа пользователей и эффективного процесса интеграции с системами пользователя сервису необходимо поддерживать основные форматы взаимодействия между сервисами:

- а) JSON;
- б) XML.

Сервис будет являться модулем, который достаточно просто встраивать в существующие системы. Для повышения удобства использования развертывание модуля должно быть простым процессом, который не занимает много времени.

Для подготовки композиций к дальнейшему анализу, а так же для их анализа, в сервисе реализованы сложные математические вычисления. Для того, чтобы повысить комфортность использования, сервису следует анализировать композиции за приемлемое время. Так же необходимо снизить требовательность к характеристикам компонентов вычислительной системы, на которой будет запущен сервис. Поэтому необходимо оптимизировать вычисления, производимые для анализа композиций.

Музыкальные жанры подвержены изменению. Для того, чтобы поддерживать аналитическую составляющую в актуальном состоянии, необходимо иметь возможность обновлять части системы. Поэтому сервис должен быть построен из независимых компонентов, согласованных на уровне интерфейсов.

Аудиозаписи могут храниться в различных форматах, поэтому необходимо реализовать поддержку обработки основных форматов распространения цифровых аудиозаписей, таких как mp3, wav, flac, m4a.

Для предоставления как можно более полной информации об аудиозаписи извлекаемые характеристики должны быть различных уровней, т.е. высокоуровневые и низкоуровневые. Определим основные характеристики композиции для извлечения:

- а) длина;
- б) громкость;
- в) количество ударов в минуту;

г) жанр.

Так же необходимо учитывать, что человеческий слух воспринимает информацию несколько иначе, чем компьютер. Поэтому нужно делать поправку на особенности человеческого слуха при анализе аудиозаписей.

### 3.2 Построение архитектуры

Сформулировав требования к программному продукту, приступим к проектированию.

Необходимо определиться с методом преобразования аудиозаписей к виду, который будет точно описывать композицию. Существует большая разница между характеристиками композиции, которые влияют на предпочтения пользователей, и характеристиками, которыми обладает сам сигнал. Поэтому работать непосредственно со звуковыми сигналами недостаточно эффективно, а так же достаточно долго. Спектрограммы ускоряют процесс работы, но так же не настолько эффективны, поскольку не учитывают особенности человеческого уха. Человеческое ухо, как говорилось ранее, воспринимает звук несколько иначе, что требует определенных корректировок. Для того, чтобы учесть поправки на восприятие звука ухом человека, можно отобразить спектрограмму на мел-шкалу. В результате мы получим мел-спектрограмму. Работать с мел-спектрограммой достаточно долго ввиду её большого размера. Необходимо найти способ сжать информацию до приемлемых размеров, при этом не потеряв в точности.

Для того, чтобы сжать информацию и учесть поправки на человеческое ухо, было решено использовать мел-частотные кепстральные коэффициенты.

Достоинства этого метода:

- а) используется спектр сигнала, что позволяет учесть волновую природу звука;
- б) спектр проецируется на мел-шкалу, что позволяет учесть восприятие частот человеческим ухом;
- в) возможность сжать количество информации количеством вычисляемых коэффициентов.

Осталось понять, как преобразовать сигнал в набор коэффициентов. Первым делом нам нужен спектр исходного сигнала, который мы получаем с помощью преобразования Фурье. Для того, чтобы не потерять временную составляющую, воспользуемся оконным преобразованием Фурье. Теперь полученный спектр нам нужно расположить на мел-шкале. Для этого мы используем окна, равномерно расположенные на мел-оси.

Если перевести график, который расположен на рисунке 3.3 в частот-

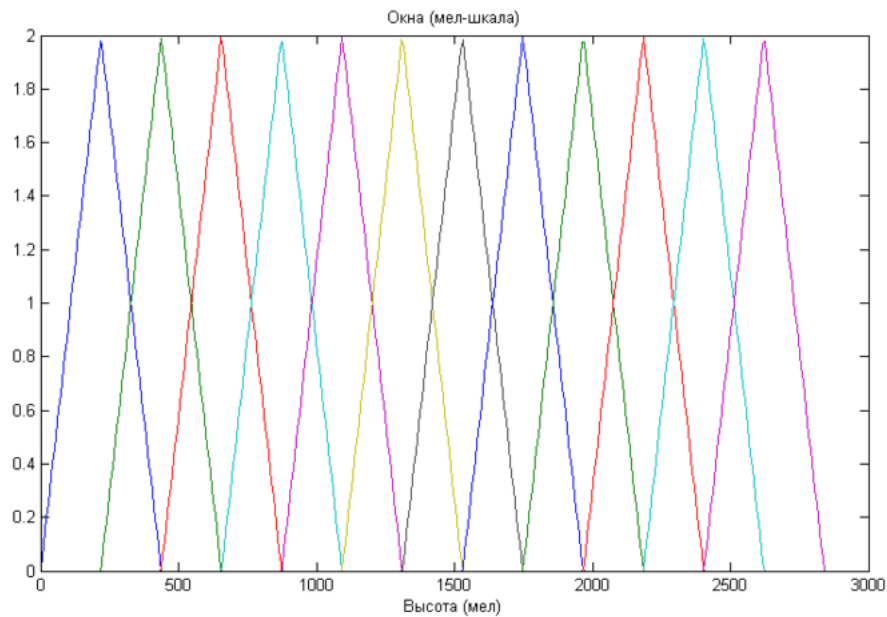


Рисунок 3.1 – Окна на мел оси

ную шкалу, можно увидеть картину, которая изображена на рисунке 3.3.

На этом графике заметно, что окна «собираются» в области низких частот, обеспечивая более высокое «разрешение» там, где оно необходимо для извлечения. Простым перемножением векторов спектра сигнала и оконной функции найдем энергию сигнала, которая попадает в каждое из окон анализа. Мы получили некоторый набор коэффициентов, но это еще не те коэффициенты, которые мы ищем. Пока их можно было бы назвать мел-частотными спектральными коэффициентами. Возводим их в квадрат и логарифмируем. Нам осталось только получить из них кепстральные, или «спектр спектра». Для этого мы могли бы еще раз применить преобразование Фурье, но лучше использовать дискретное косинусное преобразование. В результате мы получим изображение, как на рисунке 3.3.

Таким образом мы имеем очень небольшой набор значений, который при извлечении успешно заменяет тысячи отсчетов речевого сигнала.

Преобразовав сигнал, необходимо создать инструмент, который будет извлекать характеристики из полученных данных. Для этого решено использовать методы, основанные на машинном обучении. Для того, чтобы применять машинное обучение, необходимо построить модель, которая будет являться искомым инструментом. Для построения модели необходимо разработать структуру модели, а так же выбрать библиотеку для обучения.

В качестве библиотеки для обучения была выбрана TensorFlow от компании Google. Она предоставляет большой набор инструментов, при помощи которых можно обучать модели на различных системах. Так же TensorFlow



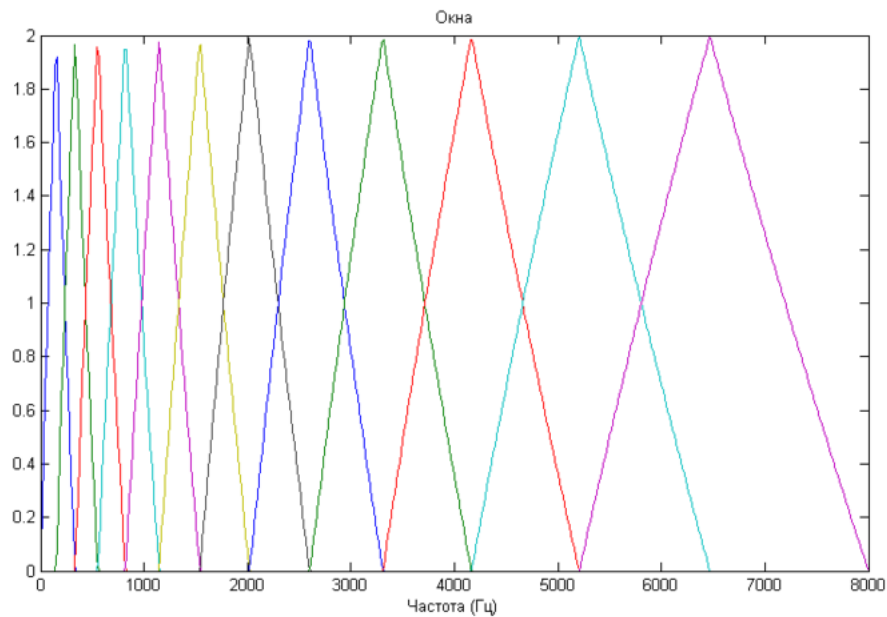


Рисунок 3.2 – Окна на мел оси, переведенные в частотную шкалу

помогает работать с высокой степенью оптимизации, что ускоряет процесс обучения и обработки данных.

Для того, чтобы ускорить процесс прототипирования и повысить уровень абстракции, поверх TensorFlow будет использоваться библиотека Keras.

Связка Keras - TensorFlow позволит оптимально расходовать ресурсы системы, на которых будет запускаться сервис, при этом будет сохранен высокий уровень абстракции, что повысит читабельность и универсальность кода.

Построим архитектуру модели, которая будет извлекать высокоуровневые характеристики из музыкальных композиций. Для извлечения признаков хорошо себя показывают глубинные нейронные сети. Для извлечения признаков из матриц больших размерностей, к которым относится полученный список мел-кепстральных коэффициентов, лучше всего использовать сверточные нейронные сети. Но для извлечения временных паттернов лучше всего себя зарекомендовали рекуррентные нейронные сети. Для получения наилучшего результата, скомбинируем сверточные нейронные сети и рекуррентные нейронные сети в глубинной нейронной сети. В результате получим нейронную сеть, в которой будет 6 слоев. Первые 4 слоя являются четырехслойной сверточной нейронной сетью, которая будет извлекать локальные признаки. Следующие 2 слоя являются двухслойной рекуррентной нейронной сетью, которая предназначена для агрегирования временных шаблонов. Это более эффективный подход, чем, например, использовать усреднение результатов.

Полученная нейронная сеть может быть больших размеров. Для того,

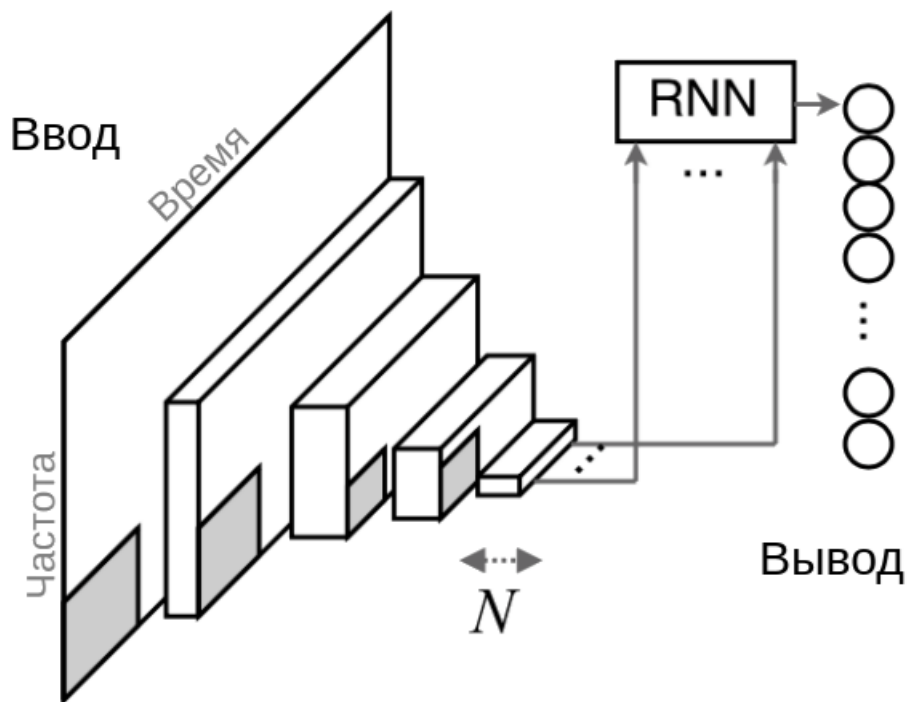


Рисунок 3.3 – Полученная структура нейронной сети

чтобы сократить время на загрузку нейронной сети из дискового пространства в память, следует держать сеть в оперативной памяти. Наиболее эффективно хранить сеть в отдельном процессе, в который будут поступать данные для обработки. Это снизит затраты по памяти, а так же позволит обрабатывать большее количество данных за единицу времени. Процесс, который хранит и использует нейронную сеть, решено реализовать в виде демона.

Для того, чтобы предоставить API, необходимо реализовать сервер, который обеспечит сетевое взаимодействие по заданному протоколу. Для реализации части сервера, которая отвечает за бизнес-логику, будет использоваться микрофреймворк Flask. Flask является микрофреймворком, что обеспечивает необходимую функциональность с потреблением минимального количества ресурсов. Эта часть сервера будет использоваться для маршрутизации запросов, сериализации и десериализации данных, а так же для обмена данными с демоном, который отвечает за обработку данных. Для того, чтобы повысить пропускную способность, часть сервера, которая отвечает за сетевое взаимодействие, будет использоваться HTTP сервер Gunicorn. Он поддерживает использование нескольких рабочих процессов для обработки запросов и одного мастер-процесса для балансировки нагрузки между рабочими процессами.

Необходимо организовать передачу данных между демон-процессом, который отвечает за обработку данных, и сервером. Для этого следует ис-

пользовать очередь задач, которая будет использоваться для последовательной обработки данных, так же необходимо использовать кеширующую систему для того, чтобы обеспечить обратное взаимодействие. В качестве очереди задач будет использоваться RedisMQ, которая использует в качестве хранилища данных базу данных Redis. В качестве кеша, который будет обеспечивать обратное взаимодействие, так же будет использоваться Redis. Преимущество Redis в данном случае заключается в том, что эта технология хранит данные в оперативной памяти, увеличивая её расход, зато это сокращает время, которое потребуется для обмена данными.

Для того, чтобы контролировать и запускать демон и сервер, следует использовать систему контроля процессов. Такой системой является Supervisor. С его помощью процессы сервера и демона будут запускаться, так же они будут перезапускаться в случае неожиданного завершения работы какого-либо из процессов. Supervisor так же позволит хранить логи приложений в указанном месте для того, чтобы обеспечить мониторинг работы системы, что поможет так же и для отладки процессов в случае их аварийного завершения.

Чтобы предоставить сервис в виде модуля, который удобно встраивать в сторонние системы, необходимо все компоненты, описанные выше, поместить в контейнер, который будет предоставляться пользователю. В качестве системы, которая обеспечит управление и развертывание контейнеров, следует использовать Docker. Преимущество Docker заключается в том, что он предоставляет неполную виртуализацию, что позволяет существенно экономить вычислительные ресурсы. Так же Docker обеспечит переносимость контейнера на другие системы, что позволит развертывать сервис с минимальными затратами. Особенности виртуализации Docker позволяют уменьшить размер контейнера, что является дополнительным удобством при переносе сервиса.

## 4 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

### 4.1 Essentia

Essentia - это библиотека C++ с открытым исходным кодом, обладающая привязками Python, предназначенная для аудиоанализа. Она выпускается под лицензией Affero GPLv3 и также доступна под собственной лицензией по запросу. Библиотека содержит обширную коллекцию многократно используемых алгоритмов, реализующих функциональность ввода и вывода звука, стандартные блоки цифровой обработки сигналов, статистическую характеристику данных и большой набор спектральных, временных, тональных музыкальных обработчиков.

Essentia - это не фреймворк, а скорее набор алгоритмов, завернутый в библиотеку. Он сконструирован с упором на надежность, производительность и оптимальность предоставляемых алгоритмов, включая скорость вычислений и использование памяти, а также простоту использования. Поток анализа определяется и реализуется пользователем, в то время как Essentia заботится о деталях реализации используемых алгоритмов. Существует специальный режим потоковой передачи, в котором возможно соединять алгоритмы и запускать их автоматически, вместо того, чтобы явно указывать порядок выполнения с преимуществом менее стандартного кода и меньшим потреблением памяти. Ряд примеров предоставляется библиотекой, однако их не следует рассматривать как единственный правильный способ делать вещи. Большая часть алгоритмов Essentia хорошо подходит для приложений реального времени.

Предоставляемые функциональные возможности легко расширяемы и позволяют проводить как исследовательские эксперименты, так и разработку крупномасштабных промышленных приложений.

### 4.2 Python

Python — высокоуровневый язык программирования общего назначения, ориентированный на повышение производительности разработчика и читаемости кода. Синтаксис ядра Python минималистичен. В то же время стандартная библиотека включает большой объём полезных функций.

Python поддерживает несколько парадигм программирования, в том числе структурное, объектно-ориентированное, функциональное, императивное и аспектно-ориентированное. Основные архитектурные черты — динамическая типизация, автоматическое управление памятью, полная интроспекция, механизм обработки исключений, поддержка многопоточных вычисле-

ний и удобные высокоуровневые структуры данных. Код в Python организуется в функции и классы, которые могут объединяться в модули (они в свою очередь могут быть объединены в пакеты).

Эталонной реализацией Python является интерпретатор CPython, поддерживающий большинство активно используемых платформ. Он распространяется под свободной лицензией Python Software Foundation License, позволяющей использовать его без ограничений в любых приложениях, включая проприетарные. Есть реализации интерпретаторов для JVM (с возможностью компиляции), MSIL (с возможностью компиляции), LLVM и других. Проект PyPy предлагает реализацию Python с использованием JIT-компиляции, которая значительно увеличивает скорость выполнения Python-программ.

Python — активно развивающийся язык программирования, новые версии (с добавлением/изменением языковых свойств) выходят примерно раз в два с половиной года. Вследствие этого и некоторых других причин на Python отсутствуют стандарт ANSI, ISO или другие официальные стандарты, их роль выполняет CPython

### 4.3 TensorFlow

TensorFlow - это библиотека программного обеспечения с открытым исходным кодом для машинного обучения по целому ряду задач и разработанная Google для удовлетворения своих потребностей в системах, способных создавать и обучать нейронные сети для обнаружения и расшифровки паттернов и корреляций, аналогичных обучению и рассуждениям, которые используют люди. В настоящее время он используется для исследований и производства продуктов Google, часто заменяя роль предшественника с закрытым исходным кодом, DistBelief. TensorFlow была первоначально разработана командой Google Brain для внутреннего использования Google, прежде чем выпустить ее под лицензией с открытым исходным кодом Apache 2.0 9 ноября 2015 года.

Начиная с 2011 года, Google Brain построила DistBelief как проприетарную систему машинного обучения, основанную на глубоком обучении нейронных сетей. Его использование быстро развивалось в различных компаниях родительской компании Alphabet как в исследовательских, так и в коммерческих целях. Google назначил нескольких компьютерных ученых, включая Джеффа Дина, для упрощения и реорганизации базы кода DistBelief в более быструю и более надежную библиотеку прикладного уровня, которая стала TensorFlow. В 2009 году команда во главе с Джефффри Хинтоном реализовала обобщенное обратное распространение и другие улучшения, которые позво-

лили создать нейронные сети с существенно большей точностью, например, на 25% сократить ошибки распознавания речи.

TensorFlow - это система машинного обучения второго поколения Google Brain, выпущенная как программное обеспечение с открытым исходным кодом 9 ноября 2015 года. Хотя эталонная реализация выполняется на отдельных устройствах, TensorFlow может работать на нескольких процессорах и графических процессорах (с дополнительными расширениями CUDA для универсальных вычислений на графических процессорах). TensorFlow доступен на 64-битных Linux, macOS и мобильных вычислительных платформах, включая Android и iOS.

Вычисления TensorFlow выражаются в виде графов прохождения данных с состоянием. Название TensorFlow происходит от операций, которые такие нейронные сети выполняют с многомерными массивами данных. Эти многомерные массивы называются тензорами. В июне 2016 года Джефф Дин из Google заявил, что 1500 репозитория на GitHub упомянули TensorFlow, из которых только 5 принадлежали Google.

В мае 2016 года Google объявил о своем тензорном процессоре (TPU) - специализированной ASIC, разработанной специально для машинного обучения и предназначенной для TensorFlow. TPU представляет собой программируемый ускоритель ИИ, предназначенный для обеспечения высокой пропускной способности арифметики с низкой точностью (например, 8-разрядной) и ориентированной на использование или запуск моделей, а не на их обучение. Google объявила, что они уже более года работают с моделями машинного обучения на основе TPU в своих центрах обработки данных, и обнаружили, что TPU обеспечивает на порядок более оптимизированную производительность на ватт для машинного обучения.

#### 4.4 Keras

Keras - это высокоуровневый API нейронных сетей, написанный на Python и способный работать поверх TensorFlow или Theano. Он был разработан с упором на быстрое проведение экспериментов. Способность идти от идеи к результату с наименьшей задержкой является ключевой особенностью данной библиотеки.

Особенности Keras:

- а) возможность легко и быстро создавать прототипы (благодаря удобству пользователя, модульности и расширяемости);
- б) поддержка как сверточных нейронных сетей, так и рекуррентных нейронных сетей, а также их комбинаций;

в) работа без проблем на процессоре и графическом процессоре.

Руководящие принципы:

а) удобство для пользователя (Keras - это API, предназначенный для людей, а не для машин, это ставит удобство использования в центр внимания и обеспечивает ясную обратную связь из-за пользовательской ошибки);

б) модульность (под моделью понимается последовательность или граф автономных, полностью конфигурируемых модулей, которые могут быть подключены вместе с минимальными ограничениями);

в) легкая расширяемость (новые модули легко добавлять, как новые классы и функции, а существующие модули предоставляют достаточно примеров, чтобы иметь возможность легко создавать новые модули);

г) работа с Python (нет отдельных конфигурационных файлов моделей в декларативном формате, модели описаны в коде Python, который является компактным, более легким для отладки и позволяет легко расширяться).

## 4.5 Flask

Flask - это небольшой фреймворк, написанный на языке Python, с весьма большим сообществом и множеством модулей на все случаи жизни. В отличие от, скажем, Django, Flask не навязывает определенное решение той или иной задачи. Вместо этого, он предлагает использовать различные сторонние или собственные решения по вашему усмотрению.

Одним из проектных решений во Flask является то, что простые задачи должны быть простыми; они не должны занимать много кода, и это не должно ограничивать вас. Поэтому было сделано несколько вариантов дизайна, некоторые люди могут посчитать это удивительным и необщепринятым. Например, Flask использует локальные треды внутри объектов, так что вы не должны передавать объекты в пределах одного запроса от функции к функции, оставаясь в безопасном треде. Хотя этот очень простой подход и позволяет сэкономить много времени, это также может вызвать некоторые проблемы для очень больших приложений, поскольку изменения в этих локальных объектах-потоках может произойти где угодно в том же потоке. Для того, чтобы решить эти проблемы, разработчики не скрывают от вас локальные объекты-потоки, вместо этого охватывают их и предоставляем вам много инструментов, чтобы сделать работу с ними настолько приятным насколько это возможно.

## 4.6 Gunicorn

Gunicorn - это HTTP-сервер Python WSGI для UNIX. Это модель рабочих процессов, перенесенная из проекта Ruby Unicorn. Сервер Gunicorn в целом совместим с различными веб-фреймворками, прост в использовании, не нагружает ресурсы сервера и является достаточно быстрым.

Особенности:

- а) поддерживает WSGI, web2py, Django и Paster;
- б) автоматическое управление рабочими процессами;
- в) простая конфигурация Python;
- г) несколько рабочих конфигураций;
- д) различные серверные триггеры для расширяемости;
- е) совместимость с Python версий 2.6 и выше, а также Python версий 3.2 и выше.

## 4.7 Supervisor

Supervisor - это система клиент-сервер, которая позволяет своим пользователям управлять несколькими процессами в UNIX-подобных операционных системах.

Часто сложно получить точный статус (запущен или не запущен) в процессах в UNIX. Файлы с идентификаторами процессов часто не соответствуют действительности. Supervisord запускает процессы как подпроцессы, поэтому он всегда знает истинный статус своих подпроцессов, и предоставляет возможность узнать статусы для его дочерних процессов.

Supervisor предоставляет вам одно место для запуска, остановки и мониторинга процессов. Процессами можно управлять индивидуально или в группах. Вы можете настроить Supervisor для предоставления локальной или удаленной командной строки и веб-интерфейса.

Supervisor запускает свои подпроцессы через fork/exec, а подпроцессы не запускает как демоны. Операционная система сразу же сигнализирует Supervisor, когда процесс завершается, в отличие от некоторых решений, которые полагаются на неточные файлы идентификаторов процессов и периодический опрос процессов, чтобы перезапустить завершившиеся процессы.

## 4.8 Docker

Docker – программное обеспечение для автоматизации развёртывания и управления приложениями в среде виртуализации на уровне операционной системы. Позволяет поместить приложение со всем его окружением и



зависимостями в контейнер, который может быть перенесен на любую Linux-систему с поддержкой cgroups в ядре, а также предоставляет среду по управлению контейнерами. Изначально использовал возможности LXC, с 2015 года применял собственную библиотеку, абстрагирующую виртуализационные возможности ядра Linux — libcontainer. С появлением Open Container Initiative начался переход от монолитной к модульной архитектуре.

Цели использования Docker:

- а) абстрагирование хост-системы от контейнеризованных приложений;
- б) простота масштабирования;
- в) простота управления зависимостями и версиями приложения;
- г) чрезвычайно легкие, изолированные среды выполнения;
- д) совместно используемые слои;
- е) возможность компоновки и предсказуемость.

Приложения, реализующие этот подход к проектированию, должны иметь следующие характеристики:

- а) они не должны полагаться на особенности хост-системы;
- б) каждый компонент должен предоставлять консистентный API, который пользователи могут использовать для доступа к сервису;
- в) каждый сервис должен принимать во внимание переменные окружения в процессе первоначальной настройки;
- г) данные приложения должны храниться вне контейнера на примонтированных томах или в отдельных контейнерах с данными.

## 5 ТРЕНИРОВКА МОДЕЛЕЙ МАШИННОГО ОБУЧЕНИЯ

В нашем случае необходимо построить классификатор, который будет создавать соответствие между музыкальной композицией и определенным музыкальным жанром. Количество жанров будет ограничено, так как набор жанров известен заранее. Для такого типа классификаторов наилучшим образом подходит обучение с учителем.

Обучение с учителем — один из способов машинного обучения, в ходе которого испытуемая система принудительно обучается с помощью примеров «стимул-реакция». С точки зрения кибернетики, является одним из видов кибернетического эксперимента. Между входами и эталонными выходами (стимул-реакция) может существовать некоторая зависимость, но она неизвестна. Известна только конечная совокупность прецедентов — пар «стимул-реакция», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость (построить модель отношений стимул-реакция, пригодных для прогнозирования), то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов, так же как и в обучении на примерах, может вводиться функционал качества.

Для того, чтобы обучить модель, необходимо обладать достаточным количеством данных для обучения. В качестве данных используются заранее размеченные композиции, т.е. композиции с известным жанром. Композиции, на которых будет обучаться модель, являются композициями типичных представителей или основателей музыкальных направлений. Всего для обучения используется около одиннадцати тысяч записей.

Для того, чтобы оптимизировать обучение, композиции были подвергнуты первичной обработке. Первичная обработка состояла в том, чтобы преобразовать композиции в векторы, которые будут поступать на вход модели. Для обработки использовался метод, который преобразовывает композицию в набор мел-частотных кепстральных коэффициентов, описанный в части 3.2.

Для обучения выборка разбивается на несколько фрагментов. Выборка разделяется на непересекающиеся подмножества элементов, на которых будет происходить обучение. В нашем случае мы будем разбивать выборку на 3 подмножества:

- а) обучающее;
- б) валидирующее;
- в) тестирующее.

Элементы обучающего подмножества будут использоваться для того, чтобы проверять реакцию модели и, соответственно, корректировать веса

так, чтобы повышать точность классификации.

Валидирующее подмножество так же используется для обучения модели. Это случаи, на которых не проверялась реакция модели. Они необходимы для того, чтобы модель не переобучилась определять жанр только для элементов обучающего подмножества. Это внесет поправку в алгоритм обучения, которая зависит только от значений метрики точности, что позволит использовать эти данные для валидации в дальнейшем.

Элементы тестирующего множества необходимы только для того, чтобы проверить итоговую точность построенных моделью предсказаний.

Обучение разделено на эпохи. Каждая эпоха состоит из нескольких шагов:

- а) обучение на обучающей выборке в прямом порядке;
- б) обучение на обучающей выборке в обратном порядке;
- в) проверка на переобучение с помощью валидирующей выборки;
- г) перемешивание элементов в обучающей выборке;
- д) перемешивание элементов в валидирующей выборке.

Перемешивание элементов необходимо для того, чтобы исключить случайный эффект, когда модель начнет переобучаться из-за того, что в нее будут поступать циклические данные.

Количество эпох для тренировки модели вычисляется эмпирически. Модели построены так, чтобы иметь возможность их дообучить при необходимости.

Для обучения моделей необходимы ресурсы, которые значительно превышают ресурсы, которые необходимы для работы моделей в обычном режиме.

## 6 РУКОВОДСТВО ПО УСТАНОВКЕ И ИСПОЛЬЗОВАНИЮ

### 6.1 Установка

Для того, чтобы установить и подготовить сервис для использования, необходимо установить Docker на используемую систему.

Далее мы опишем процесс установки Docker на системы семейства Linux/Debian.

Для начала обновим базу данных пакетов:

```
sudo apt-get update
```

Теперь установим Docker. Добавьте ключ GPG официального репозитория Docker в вашу систему:

```
sudo apt-key adv --keyserver hkp://p80.pool.sks-keyservers.  
net:80 --recv-keys 58118  
E89F3A912897C070ADBF76221572C52609D
```

Добавим репозиторий Docker в список источников пакетов утилиты АРТ:

```
sudo apt-add-repository 'deb https://apt.dockerproject.org/  
repo ubuntu-xenial main'
```

Обновим базу данных пакетов информацией о пакетах Docker из вновь добавленного репозитория:

```
sudo apt-get update
```

Далее, наконец-то, установим Docker:

```
sudo apt-get install -y docker-engine
```

После завершения выполнения этой команды Docker должен быть установлен, демон запущен, и процесс должен запускаться при загрузке системы. Проверим, что процесс запущен:

```
sudo systemctl status docker
```

Вывод должен быть похож на представленный ниже, сервис должен быть запущен и активен:

```
docker.service - Docker Application Container Engine  
Loaded: loaded (/lib/systemd/system/docker.service; enabled;  
       vendor preset: enabled)  
Active: active (running) since Sun 2016-05-01 06:53:52 CDT;  
       1 weeks 3 days ago  
Docs: https://docs.docker.com  
Main PID: 749 (docker)
```

При установке Docker мы получаем не только сервис (демон) Docker, но и утилиту командной строки docker или клиент Docker.

Далее необходимо загрузить контейнер на машину:

```
docker pull dshmatkov/genrify
```

Перед тем как загружать контейнер, необходимо авторизоваться у автора контейнера, так как он является приватным.

После нужно запустить контейнер:

```
docker run -p 127.0.0.1:80:8000 dshmatkov/genrify
```

Порт контейнера 8000 будет привязан к порту 80 по адресу 127.0.0.1 на локальной системе.

Далее можно приступить к использованию сервиса.

## 6.2 Использование

Для того, чтобы получать характеристики музыкальных композиций, необходимо пересылать файлы аудиозаписей в теле HTTP-запроса в контейнер на порт 8000.

В результате вы получите ответ от сервиса, который будет содержать искомые характеристики.

Для того, чтобы определить формат, в котором вы хотите получать ответ от сервиса, необходимо использовать заголовок Content-Type.

Для того, чтобы получать ответы в формате XML - необходимо выставить значение этого заголовка в application/xml.

Для того, чтобы получать ответы в формате JSON - необходимо выставить значение этого заголовка в application/json.

## **7 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ И ВНЕДРЕНИЯ ПРОГРАММНОГО СРЕДСТВА**

### **7.1 Характеристика программного средства**

...

Разработки проектов программных средств связана со значительными затратами ресурсов. В связи с этим создание и реализация каждого проекта программного обеспечения нуждается в соответствующем технико-экономическом обосновании [4], которое и описывается в данном разделе.

### **7.2 Определение объема и трудоемкости ПС**

Целесообразность создания ПС требует проведения предварительной экономической оценки. Экономический эффект у разработчика ПС зависит от объема инвестиций в разработку проекта, цены на готовый продукт и количества проданных копий, и проявляется в виде роста чистой прибыли.

Оценка стоимости создания ПС со стороны разработчика предполагает составление сметы затрат, вычисление цены и прибыли от реализации разрабатываемого программного средства.

Исходные данные, которые будут использоваться при расчете сметы затрат, представлены в таблице 7.1.

Перед определением сметы затрат на разработку программного средства необходимо определить его объём. Однако, на стадии ТЭО нет возможности рассчитать точные объёмы функций, вместо этого с помощью применения действующих нормативов рассчитываются прогнозные оценки.

В качестве метрики измерения объема программных средств используется строка их исходного кода (LOC – lines of code). Данная метрика широко распространена, поскольку она непосредственно связана с конечным продуктом, может применяться на всём протяжении проекта и, кроме того, может использоваться для сопоставления размеров программного обеспечения. Далее под строкой исходного кода будем понимать количество исполняемых операторов.

Расчет объема функций программного средства и общего объема приведен в таблице 7.2.

Исходя из определенной 3-ей категории сложности и общего объема ПС  $V_o = 28\,940$ , нормативная трудоемкость  $T_n = 520$  чел./д. [4, приложение 3]. Перед определением общей трудоемкости разработки необходимо определить несколько коэффициентов.

Таблица 7.1 – Исходные данные

Наименование показателя	Условное обозначение	Значение
Категория сложности		3
Дополнительный коэффициент сложности	$\sum_{i=1}^n K_i$	0,13
Степень охвата функций стандартными модулями	$K_T$	0,7
Коэффициент новизны	$K_H$	0,9
Количество дней в году	$D_r$	365
Количество праздничных дней в году	$D_{п}$	9
Количество выходных дней в году	$D_{в}$	103
Количество дней отпуска	$D_o$	21
Количество разработчиков	$Ч_p$	3
Тарифная ставка первого разряда, руб.	$T_{ч}^1$	265
Среднемесячная норма рабочего времени, ч.	$\Phi_p$	168,3
Продолжительность рабочей смены, ч.	$T_{ч}$	8
Коэффициент премирования	$K$	1,3
Норматив дополнительной заработной платы	$H_d$	20%
Норматив отчислений в ФСЗН	$H_{сз}$	34%
Норматив отчислений по обязательному страхованию	$H_{oc}$	0,6%
Норматив расходов по статье «Материалы»	$H_{мз}$	5%
Норматив расходов по статье «Машинное время»	$H_{мв}$	15%
Понижающий коэффициент к статье «Машинное время»		0,5
Стоимость машино-часа, руб.	$Ц_m$	0,8
Норматив расходов по статье «Научные командировки»	$H_{рнк}$	15%
Норматив расходов по статье «Прочие затраты»	$H_{пз}$	20%
Норматив расходов по статье «Накладные расходы»	$H_{нр}$	50%
Уровень рентабельности	$У_{рп}$	10%
Ставка НДС	$H_{дс}$	20%
Норматив затрат на освоение ПО	$H_o$	10%
Норматив затрат на сопровождение ПО	$H_c$	20%

Коэффициент сложности, который учитывает дополнительные затраты труда, связанные с обеспечением интерактивного доступа и хранения, и

Таблица 7.2 – Перечень и объём функций программного модуля

№ функции	Наименование (содержание)	Объём функции, LoC
101	Организация ввода информации	100
102	Контроль, предварительная обработка и ввод информации	500
109	Организация ввода/вывода информации в интерактивном режиме	190
111	Управление вводом/выводом	2600
204	Обработка наборов и записей базы данных	1900
207	Манипулирование данными	8000
208	Организация поиска и поиск в БД	7500
304	Обслуживание файлов	500
305	Обработка файлов	800
309	Формирование файла	1000
506	Обработка ошибочных и сбойных ситуаций	500
507	Обеспечение интерфейса между компонентами	750
601	Отладка прикладных программ в интерактивном режиме	4300
707	Графический вывод результатов	300
	Общий объем	28 940

поиска данных в сложных структурах [4, приложение 4, таблица П.4.2]

$$K_c = 1 + \sum_{i=1}^n K_i = 1 + 0,06 + 0,07 = 1,13, \quad (7.1)$$

где  $K_i$  — коэффициент, соответствующий степени повышения сложности за счет конкретной характеристики;

$n$  — количество учитываемых характеристик.

Коэффициент  $K_t$ , учитывающий степень использования при разработке стандартных модулей, для разрабатываемого приложения, в котором степень охвата планируется на уровне около 50%, примем равным 0,7 [4, приложение 4, таблица П.4.5].

Коэффициент новизны разрабатываемого программного средства  $K_n$  примем равным 0,9, так как разрабатываемое программное средство принадлежит определенному параметрическому ряду существующих программных средств [4, приложение 4, таблица П.4.4].

Исходя из выбранных коэффициентов, общая трудоемкость разработ-



ки  $T_o = T_n \cdot K_c \cdot K_t \cdot K_n = 520 \cdot 1,13 \cdot 0,7 \cdot 0,9 = 370$  чел./д.

Для расчета срока разработки проекта примем число разработчиков  $Ч_p = 3$ . Исходя из комментария к постановлению Министерства труда и социальной защиты Республики Беларусь от 05.10.16 №54 «Об установлении расчетной нормы рабочего времени на 2017 год» [5], эффективный фонд времени работы одного человека составит

$$\Phi_{эф} = D_r - D_n - D_v - D_o = 365 - 9 - 103 - 21 = 232 \text{ д.}, \quad (7.2)$$

где  $D_r$  — количество дней в году;

$D_n$  — количество праздничных дней в году;

$D_v$  — количество выходных дней в году;

$D_o$  — количество дней отпуска.

Тогда трудоемкость разработки проекта

$$T_p = \frac{T_o}{Ч_p \cdot \Phi_{эф}} = \frac{370}{3 \cdot 232} = 0,53 \text{ г.} = 194 \text{ д.} \quad (7.3)$$

Исходя из того, что разработкой будет заниматься 3 человека, можно запланировать фонд рабочего времени для каждого исполнителя

$$\Phi_{ин} = \frac{T_p}{Ч_p} = \frac{194}{3} \approx 65 \text{ д.} \quad (7.4)$$

### 7.3 Расчет сметы затрат

Основной статьей расходов на создание ПО является заработная плата разработчиков проекта. Информация об исполнителях перечислена в таблице 7.3. Кроме того, в таблице приведены данные об их тарифных разрядах, приведены разрядные коэффициенты, а также по формулам 7.5 и 7.6 рассчитаны месячный и часовой оклады.

$$T_m = T_m^1 \cdot T_k, \quad (7.5)$$

$$T_{ч} = \frac{T_m}{\Phi_p}, \quad (7.6)$$

где  $T_m$  — месячный оклад;  
 $T_m^1$  — тарифная ставка 1-го разряда (положим ее равной 265 руб.);  
 $T_k$  — тарифный коэффициент;  
 $T_{\text{ч}}$  — часовой оклад;  
 $\Phi_p$  — среднемесячная норма рабочего времени (в 2017 г. составляет 168,3 ч. [5]).

Таблица 7.3 – Работники, занятые в проекте

Исполнители	Разряд	Тарифный коэффициент	Месячный оклад, руб.	Часовой оклад, руб.
Руководитель проекта	17	3,98	1054,70	6,27
Ведущий инженер-программист	15	3,48	922,20	5,48
Инженер-программист II категории	11	2,65	702,25	4,17

Тогда основная заработная плата исполнителей составит

$$Z_o = \sum_{i=1}^n T_{\text{чи}} \cdot T_{\text{ч}} \cdot \Phi_{\text{пи}} \cdot K =$$

$$= (6,27 + 5,48 + 4,17) \cdot 8 \cdot 65 \cdot 1,3 = 10\,761,92 \text{ руб.}, \quad (7.7)$$

где  $T_{\text{чи}}$  — часовая тарифная ставка  $i$ -го исполнителя, руб.;  
 $T_{\text{ч}}$  — количество часов работы в день;  
 $\Phi_{\text{пи}}$  — плановый фонд рабочего времени  $i$ -го исполнителя, д.;  
 $K$  — коэффициент премирования (принятый равным 1,3).

Дополнительная заработная плата 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 нормативу, установленному в организации, в процентах к основной заработной плате. Приняв данный норматив  $H_d = 20\%$ , рассчитаем дополнительные выплаты

$$Z_d = \frac{Z_o \cdot H_d}{100\%} = \frac{10\,761,92 \cdot 20\%}{100\%} = 2152,38 \text{ руб.} \quad (7.8)$$

Отчисления в фонд социальной защиты населения и в фонд обязательного страхования определяются в соответствии с действующим законодательством по нормативу в процентном отношении к фонду основной и до-

полнительной зарплат по следующим формулам

$$\begin{aligned} Z_{сз} &= \frac{(Z_o + Z_d) \cdot H_{сз}}{100\%}, \\ Z_{ос} &= \frac{(Z_o + Z_d) \cdot H_{ос}}{100\%}. \end{aligned} \quad (7.9)$$

В настоящее время нормы отчислений в ФСЗН  $H_{сз} = 34\%$  и в фонд обязательного страхования  $H_{ос} = 0,6\%$ . Исходя из этого, размеры отчислений

$$\begin{aligned} Z_{сз} &= \frac{(10\,761,92 + 2152,38) \cdot 34\%}{100\%} = 4390,86 \text{ руб.}, \\ Z_{ос} &= \frac{(10\,761,92 + 2152,38) \cdot 0,6\%}{100\%} = 77,49 \text{ руб.} \end{aligned} \quad (7.10)$$

Расходы по статье «Материалы» отражают траты на магнитные носители, бумагу, красящие материалы, необходимые для разработки ПО определяются по нормативу к фонду основной заработной платы разработчиков. Исходя из принятого норматива  $H_{мз} = 5\%$  определим величину расходов

$$M = \frac{Z_o \cdot H_{мз}}{100\%} = \frac{10\,761,92 \cdot 5\%}{100\%} = 538,10 \text{ руб.} \quad (7.11)$$

Расходы по статье «Машинное время» включают оплату машинного времени, необходимого для разработки и отладки ПО, которое определяется по нормативам на 100 строк исходного кода. Норматив зависит от характера решаемых задач и типа ПК; для текущего проекта примем  $H_{мв} = 15\%$  [4, приложение 6]. Примем величину стоимости машино-часа  $C_m = 0,8$  руб. Тогда, применяя понижающий коэффициент 0,5, получим величину расходов

$$P_m = C_m \cdot \frac{V_o}{100} \cdot H_{мв} = 0,8 \cdot \frac{28\,940}{100} \cdot 15\% \cdot 0,5 = 1736,40 \text{ руб.} \quad (7.12)$$

Расходы по статье «Научные командировки» определяются по нормативу в процентах к основной заработной плате. Принимая норматив равным  $H_{рнк} = 15\%$  получим величину расходов

$$P_{рнк} = \frac{Z_o \cdot H_{рнк}}{100\%} = \frac{10\,761,92 \cdot 15\%}{100\%} = 1614,29 \text{ руб.} \quad (7.13)$$

Расходы по статье «Прочие затраты» включают затраты на приобретение и подготовку специальной научно-технической информации и специальной литературы. Определяются по нормативу в процентах к основной зара-

ботной плате. Принимая норматив равным  $H_{пз} = 20\%$  получим величину расходов

$$П_з = \frac{З_o \cdot H_{пз}}{100\%} = \frac{10\,761,92 \cdot 20\%}{100\%} = 2152,38 \text{ руб.} \quad (7.14)$$

Затраты по статье «Накладные расходы», связанные с необходимостью содержания аппарата управления, вспомогательных хозяйств и опытных (экспериментальных) производств, а также с расходами на общехозяйственные нужды, относятся к конкретному ПО по нормативу в процентном отношении к основной заработной плате исполнителей. Принимая норматив равным  $H_{нр} = 50\%$  получим величину расходов

$$P_n = \frac{З_o \cdot H_{нр}}{100\%} = \frac{10\,761,92 \cdot 50\%}{100\%} = 5380,96 \text{ руб.} \quad (7.15)$$

Общая сумма расходов по смете определяется как сумма вышерассчитанных показателей

$$C_{п} = З_o + З_d + З_{сз} + З_{ос} + M + P_m + P_{нк} + П_з + P_n = 28\,804,78 \text{ руб.} \quad (7.16)$$

Рентабельность определяется из результатов анализа рыночных условий и переговоров с потребителями ПО. Исходя из принятого уровня рентабельности  $Y_{рп} = 10\%$ , прибыль от реализации ПО составит

$$П_o = \frac{C_{п} \cdot Y_{рп}}{100\%} = \frac{28\,804,78 \cdot 10\%}{100\%} = 2880,48 \text{ руб.} \quad (7.17)$$

На основании расчета прибыли и уровня себестоимости рассчитаем прогнозируемую цену программного средства без учета налогов

$$Ц_{п} = C_{п} + П_o = 28\,804,78 + 2880,48 = 31\,685,26 \text{ руб.} \quad (7.18)$$

Далее рассчитаем налог на добавленную стоимость

$$НДС = \frac{Ц_{п} \cdot H_{дс}}{100\%} = \frac{31\,685,26 \cdot 20\%}{100\%} = 6337,05 \text{ руб.} \quad (7.19)$$

НДС включается в прогнозируемую отпускную цену

$$Ц_o = Ц_{п} + НДС = 31\,685,26 + 6337,05 = 38\,022,31 \text{ руб.} \quad (7.20)$$

Организация-разработчик участвует в освоении и внедрении ПО и несет соответствующие затраты, которые определяются по нормативу  $H_o = 10\%$

от себестоимости ПО в расчете на три месяца

$$P_o = \frac{C_n \cdot H_o}{100\%} = \frac{28\,804,78 \cdot 10\%}{100\%} = 2880,48 \text{ руб.} \quad (7.21)$$

Кроме того, организация-разработчик осуществляет сопровождение ПО,<sup>1)</sup> которое также оплачивается заказчиком. Расчет осуществляется в соответствии с нормативом  $H_c = 20\%$  от себестоимости ПО

$$P_c = \frac{C_n \cdot H_c}{100\%} = \frac{28\,804,78 \cdot 20\%}{100\%} = 5760,96 \text{ руб.} \quad (7.22)$$

Экономическим эффектом разработчика будет являться сумма прибыли с вычетом налога на прибыль

$$P_q = P_o - \frac{P_o \cdot H_n}{100\%} = 2880,48 - \frac{2880,48 \cdot 18\%}{100\%} = 2361,99 \text{ руб.} \quad (7.23)$$

#### 7.4 Оценка экономической эффективности применения ПС у пользователя

В результате применения нового ПО пользователь может понести значительные капитальные затраты на приобретение и освоение ПО, доукомплектованием ЭВМ новыми техническими средствами и пополнение оборотных средств. Однако, если приобретенное ПО будет в достаточной степени эффективнее базового, то дополнительные капитальные затраты быстро окупятся.

Для определения экономического эффекта от использования нового ПО у потребителя необходимо сравнить расходы по всем основным статьям сметы затрат на эксплуатацию нового ПО с расходами по соответствующим статьям базового варианта. При этом за базовый вариант примем ручной вариант. Исходные данные для расчета приведены в таблице 7.4.

Общие капитальные вложения заказчика (потребителя) вычисляются следующим образом

$$K_o = K_{пр} + K_{ос} + K_c = 38\,022,31 + 2880,48 + 5760,96 = 46\,663,75 \text{ руб.} \quad (7.24)$$

где  $K_{пр}$  — затраты пользователя на приобретение ПО по отпускной цене;  
 $K_{ос}$  — затраты пользователя на освоение;  
 $K_c$  — затраты пользователя на оплату услуг по сопровождению.

---

<sup>1)</sup>На февраль 2017 г. [6]

Таблица 7.4 – Исходные данные

Наименование показателя	Условное обозначение	Значение в базовом варианте	Значение в новом варианте
Затраты пользователя на приобретение ПО	$K_{пр}$	—	38 022,31 руб.
Затраты пользователя на освоение	$K_{ос}$	—	2880,48 руб.
Затраты пользователя на сопровождение	$K_c$	—	5760,96 руб.
Трудоемкость на задачу, чел./ч.	$T_{c1}, T_{c2}$	2	0,1
Средняя зарплата <sup>1)</sup> , руб.	$Z_{см}$	716,5	716,5
Количество выполняемых задач	$A_1, A_2$	1460	1460
Время простоя сервиса, мин. в день	$P_1, P_2$	50	10
Стоимость одного часа простоя, руб.	$C_{п}$	79,8	79,8

В качестве типичного примера использования разрабатываемого ПС предполагается сценарий ее использования ежедневно 4 раза в день, при этом предполагается снижение трудоемкости с  $T_{c1} = 2$  чел./ч. до  $T_{c2} = 0,1$  чел./ч.. Приняв среднемесячную заработную плату работника  $Z_{см} = 716,5$  руб., рассчитаем экономию затрат на заработную плату в расчете на одну задачу  $C_{зе}$  и за год  $C_3$

$$C_{зе} = \frac{Z_{см} \cdot (T_{c1} - T_{c2})}{\Phi_p} = \frac{716,5 \cdot (2 - 0,1)}{168,3} = 8,09 \text{ руб.}, \quad (7.25)$$

$$C_3 = C_{зе} \cdot A_2 = 8,09 \cdot 1460 = 11\,811,4 \text{ руб.}, \quad (7.26)$$

где  $\Phi_p$  — среднемесячная норма рабочего времени, ч.

Экономия с учетом начислений на зарплату

$$C_n = C_3 \cdot \frac{100\% + K}{100\%} = 11\,811,4 \cdot \frac{100\% + 1,3}{100\%} = 11\,964,95 \text{ руб.}, \quad (7.27)$$

где  $K$  — норматив начислений на зарплату, руб.

Экономия за счет сокращения простоев сервиса

$$C_c = \frac{(П_1 - П_2) \cdot Д_{\text{рг}} \cdot C_{\text{п}}}{60} = \frac{(50 - 10) \cdot 300 \cdot 79,8}{60} = 15\,960,00, \text{ руб.}, \quad (7.28)$$

где  $Д_{\text{рг}}$  — плановый фонд работы сервиса, д.

Тогда общая годовая экономия текущих затрат, связанных с использованием нового ПО

$$C_o = C_{\text{н}} + C_c = 11\,964,95 + 15\,960,00 = 27\,924,95 \text{ руб.} \quad (7.29)$$

Внедрение нового ПО позволит пользователю сэкономить на текущих затратах, то есть получить на эту сумму дополнительную прибыль. Для пользователя в качестве экономического эффекта выступает лишь чистая прибыль. Принимая размер ставки налога на прибыль  $Н_{\text{п}} = 18\%$  получим

$$\Delta П_{\text{ч}} = C_o - \frac{C_o \cdot Н_{\text{п}}}{100\%} = 27\,924,95 - \frac{27\,924,95 \cdot 18\%}{100\%} = 22\,898,46 \text{ руб.} \quad (7.30)$$

В процессе использования нового ПО чистая прибыль в конечном итоге возмещает капитальные затраты. Однако полученные при этом суммы прибыли и затрат по годам приводят к единому времени – расчетному году (за расчетный год принят 2017-й год) путем умножения результатов и затрат за каждый год на коэффициент дисконтирования  $\alpha$ . При расчете используются следующие коэффициенты: 2017 г. – 1, 2018 г. – 0,8696, 2019 г. – 0,7561, 2020 г. – 0,6575. Все рассчитанные данные экономического эффекта сводятся в таблицу 7.5.

В результате технико-экономического обоснования применения программного средства были получены следующие значения показателей эффективности:

- чистая прибыль разработчика составит  $П_{\text{ч}} = 2361,99$  руб.;
- затраты заказчика окупятся уже на четвертом году использования;
- экономическая эффективность для заказчика, выраженная в виде чистого дисконтированного дохода, составит 5618,02 руб. за четыре года использования данного ПС; более высокий прирост прибыли заказчик получит по истечению данного срока.

Полученные результаты свидетельствуют об эффективности разработки и внедрения проектируемого программного средства.

Таблица 7.5 – Расчет экономического эффекта от использования нового ПО

Наименование показателя	2017 г.	2018 г.	2019 г.	2020 г.
<i>Результаты</i>				
Коэффициент приведения	1	0,8696	0,7561	0,6575
Прирост прибыли за счет экономии затрат (П <sub>ч</sub> ), руб.	—	22 898,46	22 898,46	22 898,46
То же с учетом фактора времени, руб.	—	19 912,50	17 313,53	15 055,74
<i>Затраты</i>				
Приобретение ПО (К <sub>пр</sub> ), руб.	38 022,31	—	—	—
Освоение ПО (К <sub>ос</sub> ), руб.	2880,48	—	—	—
Сопровождение ПО (К <sub>с</sub> ), руб.	5760,96	—	—	—
Всего затрат (К <sub>о</sub> ), руб.	46 663,75	—	—	—
То же с учетом фактора времени, руб.	46 663,75	—	—	—
<i>Экономический эффект</i>				
Превышение результата над затратами, руб.	–46 663,75	19 912,5	17 313,53	15 055,74
То же с нарастающим итогом, руб.	–46 663,75	–26 751,25	–9437,72	5618,02



## **ЗАКЛЮЧЕНИЕ**

Данный дипломный проект относится к области извлечения музыкальной информации. Был проведен анализ продуктов, которые являются аналогами реализуемого проекта, а так же предметной области в целом. По результатам анализа был сделан вывод, что на данный момент в этой области ведутся активные исследования и разработки, так же были оценены аналоги, выявлены их достоинства и недостатки.

На основании проведенного анализа предметной области были выдвинуты требования к программному средству. В качестве технологий разработки были выбраны наиболее современные средства, применяемые для решения подобных задач. Спроектированное средство показало высокую точность классификации композиций по жанрам, что показывает эффективность выбранного подхода. Расширение распространения автоматизации на извлечение музыкальных характеристик высокого уровня позволяет сделать вывод о целесообразности разработки данного программного средства. Это было подтверждено так же в ходе выполнения экономического обоснования.

В итоге было разработано программное средство, позволяющее автоматизированно извлекать характеристики музыкальных композиций высокого уровня.

В дальнейшем планируется увеличивать точность извлечения характеристик. Так же планируется применить подходы, которые не требуют тренировки и способны самостоятельно обучаться извлекать характеристики в процессе работы. Кроме извлекаемых, существуют другие характеристики композиций, поддержку которых планируется внедрить в данное программное средство.

## СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

[1] van der Maaten, L.J.P. Visualizing High-Dimensional Data Using t-SNE / L.J.P. van der Maaten, G.E. Hinton. — Journal of Machine Learning Research, 2008.

[2] van den Oord, Aaron. Deep content-based music recommendation / Aaron van den Oord, Sander Dieleman, Benjamin Schrauwen. — Electronics and Information Systems department (ELIS), Ghent University. — <http://papers.nips.cc/paper/5004-deep-content-based-music-recommendation.pdf>.

[3] Buduma, Nikhil. Fundamentals of Deep Learning / Nikhil Buduma. — O'Reilly Media, 2017.

[4] Палицын, В.А. Техничко-экономическое обоснование дипломных проектов: Метод. пособие для студ. всех спец. БГУИР. В 4-х ч. Ч. 4: Проекты программного обеспечения / В.А. Палицын. — Мн : БГУИР, 2006. — 76 с.

[5] Пещенко, Е.А. Производственный календарь на 2017 год [Электронный ресурс]. — Электронные данные. — Режим доступа: <http://www.mintrud.gov.by/system/extensions/spaw/uploads/files/Kommetarij-2017-RV.pdf>. — Дата доступа: 06.04.17.

[6] Белстат. О начисленной средней заработной плате работников в феврале 2017 г. [Электронный ресурс]. — Электронные данные. — Режим доступа: [http://www.belstat.gov.by/ofitsialnaya-statistika/solialnaya-sfera/trud/operativnaya-informatsiya\\_8/o-nachislennoi-srednei-zarabotnoi-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-v-fevrале-2017-nbsp-g/](http://www.belstat.gov.by/ofitsialnaya-statistika/solialnaya-sfera/trud/operativnaya-informatsiya_8/o-nachislennoi-srednei-zarabotnoi-plate-rabotnikov/o-nachislennoy-sredney-zarabotnoy-plate-rabotnikov-v-fevrале-2017-nbsp-g/). — Дата доступа: 07.04.17.

# ПРИЛОЖЕНИЕ А

## (обязательное)

### Фрагменты исходного кода

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8" />
  <link rel="stylesheet" href="./node_modules/semantic-ui-css/semantic.min.css"/>
</head>
<body>
  <div id="root"></div>
  <script src="./node_modules/react/dist/react.js"></script>
  <script src="./node_modules/react-dom/dist/react-dom.js"></script>

  <script src="./dist/bundle.js"></script>
</body>
</html>
```