Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «**Информатика и системы управления**»

КАФЕДРА «**Программное обеспечение ЭВМ и информационные технологии**»

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ НА ТЕМУ:

Метод выделения звуков естественного языка в звучащей речи.

Студент группы **ИУ7-82Б** _____ **Левушкин И.К.**
(Подпись, дата) (И.О. Фамилия)

Руководитель ВКР _____ **Градов В.М.**
(Подпись, дата) (И.О. Фамилия)

Консультант _____ **Строганов Ю.В.**
(Подпись, дата) (И.О. Фамилия)

Нормоконтроллер _____ **Строганов Ю.В.**
(Подпись, дата) (И.О. Фамилия)

***2021** г.*

РЕФЕРАТ

Расчетно-пояснительная записка 64 с., 22 рис., 6 табл., 27 ист., 1 прил.
РАСПОЗНАВАНИЕ РЕЧИ, СЕГМЕНТАЦИЯ РЕЧЕВОГО СИГНАЛА, ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЕ, МЕТОД ОПОРНЫХ ВЕКТОРОВ

Объектом исследования являются системы распознавания речи. Предметом исследования - методы автоматической контекстно-независимой временной сегментации речевого сигнала. Цель работы - разработка метода выделения звуков естественного языка в звучащей речи. Задачи, решаемые в работе:

- анализ предметной области - описание основных этапов распознавания речевого сигнала, обоснование актуальности задачи сегментации речевого сигнала, проведение анализа существующих решений сегментации речевого сигнала;
- проведение анализа существующих методов выделения признаков речевого сигнала и алгоритмов классификации;
- выбор методов наиболее оптимальных для поставленной цели;
- проектирование метода выделения звуков естественного языка в звучащей речи, разработка алгоритма, реализующего данный метод;
- проектирование системы, проверяющей работоспособность метода;
- формирование выборки для проведения эксперимента;
- проведение исследования разработанного метода на данной выборке.

Область применения - системы распознавания речи. Предлагаемые направления развития:

- подготовка большего количества обучающей выборки, имеющей меньшую погрешность измерений для повышения точности метода;
- применеие альтернативных алгоритмов классификации, позволяющих повысить качество распознавания с применением большего количества данных.

Поставленная цель была достигнута: метод выделения звуков естественного языка в звучащей речи был разработан, реализован и протестирован. Были рассмотрены существующие недостатки метода и предложены пути дальнейшего развития.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Распознавание речи — автоматический процесс преобразования речевого сигнала в цифровую информацию, является одной из актуальных и сложных задач настоящего времени, имеющая множество успешных применений в различных сферах бизнеса таких как:

- Телефония: автоматизация обработки входящих и исходящих звонков [2];
- Бытовая техника: голосовое управление бытовой техникой [2];
- Десктопы, ноутбуки и мобильные устройства: голосовой ввод в играх и приложениях [2];
- Автомобили: голосовое управление в салоне автомобиля [2];
- Социальные сервисы для людей с ограниченными возможностями [3].

Здесь задействованы достижения весьма различных областей: от компьютерной лингвистики до цифровой обработки сигналов.

Несмотря на значительный прогресс в точности и скорости современных систем, их результаты все еще заметно далеки от желаемых, поскольку человеческая речь представляет собой сложный для распознавания нестационарный нелинейный сигнал.

В системах распознавания речи первоочередной задачей, требующей решения, является задача автоматической сегментации в соответствии с фонетической транскрипцией языка. Под сегментацией понимается операция выделения звуков естественного языка (основных лингвистических элементов языка [1]) в звучащей речи.

Целью работы является разработка метода выделения звуков естественного языка в звучащей речи. Для достижения поставленной цели необходимо решить следующие **задачи**:

- провести анализ предметной области:
  - описать основные этапы распознавания речевого сигнала;
  - обосновать актуальность задачи сегментации речевого сигнала;
  - провести анализ существующих решений сегментации речевого сигнала.
- провести анализ существующих методов выделения признаков речевого сигнала и алгоритмов классификации;

- выбрать методы наиболее оптимальные для поставленной цели;
- на их основе спроектировать метод выделения звуков естественного языка в звучащей речи и разработать алгоритм, реализующий данный метод;
- спроектировать систему для проверки работоспособности метода;
- сформировать выборку для проведения эксперимента;
- провести исследование разработанного метода на данной выборке;

1 Аналитическая часть

В данном разделе проведен анализ предметной области - описаны основные этапы распознавания речевого сигнала, дано обоснование актуальности задачи сегментации речевого сигнала, проведен анализ существующих решений сегментации речевого сигнала.

Проанализированы существующие методы выделения признаков речевого сигнала и алгоритмов классификации.

Объектом исследования являются системы распознавания речи. **Предметом исследования** - методы автоматической контекстно-независимой временной сегментации речевого сигнала.

1.1 Анализ предметной области

Распознавание речи — процесс преобразования речевого сигнала в цифровую информацию.

Задачей распознавания является сопоставление набору акустических признаков речевого сигнала или наблюдений $X(x_1, ..., x_n)$ последовательности слов $W(w_1, ..., w_k)$, имеющих наибольшую вероятность правдоподобия среди всех кандидатов. Для этого используется формула Байеса:

$$W = argmax \left[\frac{P(W) * P(X|W)}{P(X)} \right] \tag{1}$$

Причем, в процессе распознавания вероятность уже полученных признаков P(X) не подлежит оптимизации и знаменатель в формуле не испльзуется:

$$W = argmax \left[P(W) * P(X|W) \right] \tag{2}$$

1.1.1 Структура систем распознавания речи

Системы распознавания речи впервые появились в 1952 году. С тех пор методы распознавания не раз менялись. Ранее использовались такие методы и алгоритмы, как:

- Динамическое программирование (Dynamic Time Warping) - временные динамические алгоритмы, выполняющие классификацию на основе сравнения с эталоном;
- Методы дискриминантного анализа, основанные на Байесовской дискриминации (Bayesian discrimination);

- Скрытые Марковские Модели (Hidden Markov Model).
- Нейронные сети (Neural Networks).

В настоящее время, перечисленные выше методы как правило комбинируются. Их сочетание позволяет получить более высокое качество распознавания, чем использование каждой модели отдельно.

Системы распознавания речи имеют следующие основные модули [4], [5]:

- Акустическая модель - функция, принимающая на вход признаки на небольшом участке акустического сигнала (фрейме) и выдающая распределение вероятностей различных фонем на этом фрейме. Таким образом, акустическая модель дает возможность по звуку восстановить, что было произнесено — с той или иной степенью уверенности.;
- Языковая модель - позволяет узнать, какие последовательности слов в языке более вероятны, а какие менее. Здесь в самом простом случае требуется предсказать следующее слово по известным предыдущим словам;
- Декодер - компонент, который сопоставляет входной речевой поток с информацией, хранящейся в акустических и языковых моделях, и определяет наиболее вероятную последовательность слов, которая и является конечным результатом распознавания.

1.1.2 Сегментация речевого сигнала

Для эффективной работы системы акустическая модель должна получать на вход признаки, которые бы наилучшим образом описывали особенности данного сигнала. Но даже при наличии возможности использовать наиболее информативные методы выделения признаков речевого сигнала, результат распознавания зачастую далек от совершенстваиз-за сложной структуры речевого сигнала, недостаточной изученности механизмов речеобразования и речевосприятия.

Чтобы повысить точность акустической модели, помимо основных признаков необходимо также подавать на вход «метапризнаки» сигнала.

Одними из таких признаков могут служить границы между основными лингвистическими элементами языка. Другими словами, возникает потребность в решении задачи временной сегментации речевого сигнала.

Рассмотрев существующие решения сегментации речевого сигнала ([7],

[8], [9]) были выделены два основных этапа в задаче сегментации:

- **выделение признаков сигнала**;
- **определение межфонемных переходов на основе выделенных признаков сигнала**.

Существует два основных типа алгоритмов сегментации речи.

- **Контекстно-зависимая сегментация** - к этому типу относятся алгоритмы, которые производят сегментацию речи при условии, что известна последовательность фонем данной фразы;
- **Контекстно-независимая сегментация** - другой тип алгоритмов, не использующий априорной информации о фразе, при этом границы сегментов определяются по степени изменения акустических характеристик сигнала.

При автоматической сегментации желательно использовать только общие характеристики речевого сигнала, поскольку обычно на этом этапе нет конкретной информации о содержании речевого высказывания, поэтому в данной работе **предметом исследования** являются методы автоматической контекстно-независимой временной сегментации речевого сигнала.

1.2 Методы выделения признаков речевого сигнала

Сегментация речевого сигнала подразумевает выделение участков сигнала, соответствующим отдельным структурным единицам. Для решения данной задачи необходимо выбрать метод выделения акустических характеристик сигнала на заданных интервалах времени. На основе этих признаков можно будет оценить степень изменения сигнала во времени и таким образом определить границы между основными лингвистическими элементами языка.

Ниже приведен анализ наиболее распространенных методов выделения признаков речевого сигнала.

1.2.1 Преобразование Фурье

Преобразование Фурье — это функция, описывающая амплитуду и фазу каждой синусоиды, соответствующей определённой частоте, где амплитуда представляет собой высоту кривой, а фаза - начальную точку синусоиды [10]. Эта новая функция описывает коэффициенты при разложении исходной функции на элементарные составляющие — гармонические колебания с разными частотами (подобно тому, как музыкальный аккорд может быть выражен в виде амплитуд нот, которые его составляют). Преобразование Фурье функции f

вещественной переменной является интегральным и задаётся с помощью следующей формулы:

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) e^{-it\omega}\, dt, \tag{3}$$

где

- $f(t)$ - исходный сигнал;
- $F(t)$ - Фурье-образ функции $f(t)$ или частотный спектр функции $f(t)$;
- ω - круговая частота.

1.2.2 Оконное преобразование Фурье

Классическое преобразование Фурье имеет дело со спектром сигнала, взятым во всем диапазоне существования переменной. Нередко интерес представляет только локальное распределение частот, в то время как требуется сохранить изначальную переменную (обычно время).

С позиций точного представления произвольных сигналов и функций, преобразование Фурье имеет ряд недостатков, которые привели к появлению оконного преобразования Фурье и стимулировали развитие вейвлет-преобразования [11]:

- ограниченная информативность анализа нестационарных сигналов и практически полное отсутствие возможностей анализа их особенностей (сингулярностей), так как в частотной области происходит «размазывание» особенностей сигналов (разрывов, ступеней, пиков и т. п.) по всему частотному диапазону спектра;
- появление эффекта Гиббса на скачках функций, при усечениях сигналов и при вырезке отрезков сигналов для локального детального анализа;
- гармонический характер базисных функций, определенных в интервале от $-\infty$ до $+\infty$.

Неспособность преобразования Фурье осуществлять временную локализацию сингулярностей сигналов может быть частично устранена введением в преобразование так называемой движущейся оконной функции, имеющей компактный носитель. Использование оконной функции позволяет представлять результат преобразования в виде функции двух переменных — частоты и временного положения окна.

Оконное преобразование Фурье имеет следующий вид [12]:

$$F(t,\ \omega) = \int\limits_{-\infty}^{\infty} f(\tau) W(\tau - t) e^{-i\omega\tau}\, d\tau, \tag{4}$$

где

- $F(t,\ \omega)$ даёт распределение частот части оригинального сигнала $f(t)$ в окрестности момента времени t;
- $W(\tau - t)$ - некоторая оконная функция.

1.2.3 Вейвлет-преобразование

Широко используемое преобразование Фурье для анализа сигналов, как непрерывных, так и дискретных, оказывается недостаточно эффективным при обработке сложных сигналов.

Например, Фурье спектры для сигналов из двух синусоид, которые с разными частотами, первый из которых, представляющий собой сумму синусоид, а второй представляет собой, последовательно следующие друг за другом синусоиды, одинаковы и будут выглядеть как два пика на двух фиксированных частотах (рисунок 1). Из этого следует, преобразование Фурье в своём обычном виде не приспособлено для анализа нестационарных сигналов, так как теряется информация о временных характеристиках сигнала.

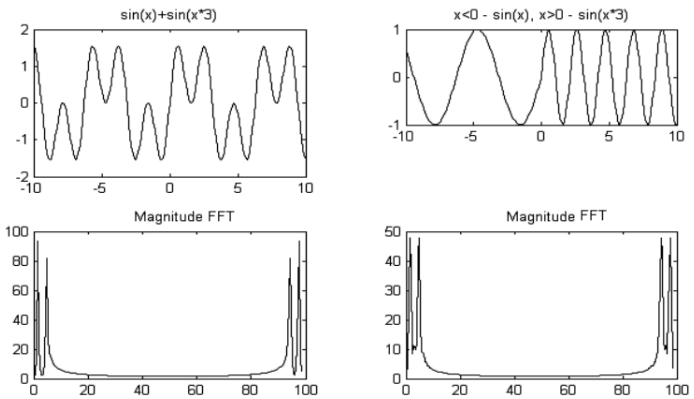

Рисунок 1 – Пример неинформативности преобразования Фурье

Речевой сигнал является примером нестационарного процесса, в котором информативным является сам факт изменения его частотно-временных характеристик.

Для анализа таких процессов требуются базисные функции, способные выявлять в исследуемом сигнале как частотные, так и его временные характеристики, то есть функции со свойствами частотно-временной локализации. Такие возможности предоставляют вейвлеты, являющиеся обобщением спектрального анализа [10].

Ниже приведена общая формула вейвлет-преобразования функции $f(t)$.

$$W_\Psi(x,a) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(t)\Psi(\frac{t-x}{a})dt, \tag{5}$$

где

- $\Psi(\frac{t-x}{a})$ - вейвлет;
- a - масштабный коэффициент;
- x - параметры сдвига.

Вейвлеты – функции двух аргументов – масштаба и сдвига. В отличие от стандартного преобразования Фурье, они позволяют обрабатывать сигнал одновременно в физическом – время, координата; и частотном пространствах.

Выбор анализирующего вейвлета определяется тем, какую информацию необходимо извлечь из сигнала. Каждый вейвлет имеет характерные особенности во временном и в частотном пространстве, поэтому иногда с помощью разных вейвлетов можно полнее выявить и подчеркнуть те или иные свойства анализируемого сигнала.

Таким образом, вейвлет-преобразование обеспечивает двумерное представление исследуемого сигнала в частотной области в плоскости частота-положение. Аналогом частоты при этом является масштаб аргумента базисной функции (чаще всего – времени), а положение характеризуется её сдвигом. Это позволяет найти особенности сигналов, одновременно локализуя их на временной шкале. Другими словами, вейвлет-анализ можно охарактеризовать как спектральный анализ локальных возмущений.

Используя вейвлет-преобразование, сигнал можно представить как последовательность образов с разной степенью детализации, что позволяет найти локальные особенности сигнала и классифицировать их по интенсивности.

Многомасштабный Вейвлет-анализ основывается на разложении сигнала по функциям, образующим ортонормированный базис. Каждую функцию можно разложить на некотором заданном уровне разрешения (масштабе) j_n в

ряд вида [10]:

$$f(x) = \sum_{k=0}^{2M-1} s_{j_n,k}\varphi_{j_n,k} + \sum_{j\geq j_n}^{j_{max}} \sum_{k=0}^{2M-1} d_{j,k}\Psi_{j,k}, \qquad (6)$$

где

- $\varphi_{j_n,k}$ и $\Psi_{j,k}$ - масштабированные и смещенные версии скейлинг-функции (масштабной функции) φ и материнского вейвлета Ψ;
- $s_{j,k}$ - коэффициенты аппроксимации;
- $d_{j,k}$ - детализирующие коэффициенты.

На рисунках 2 - 5 представлены аналитические графики функций φ и Ψ указанных вейвлетов.

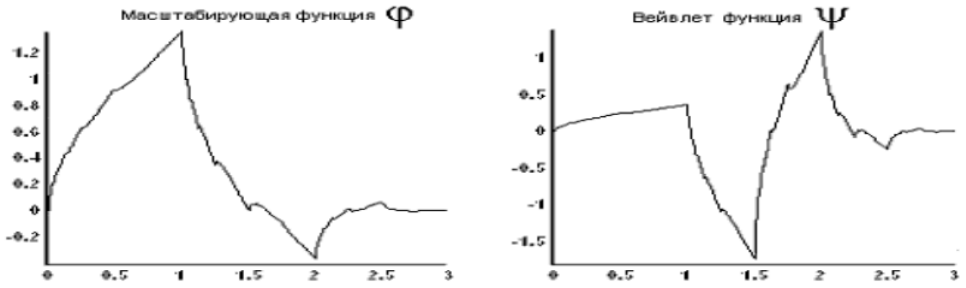

Рисунок 2 – Аналитические графики функций φ и Ψ, вейвлет Симлета

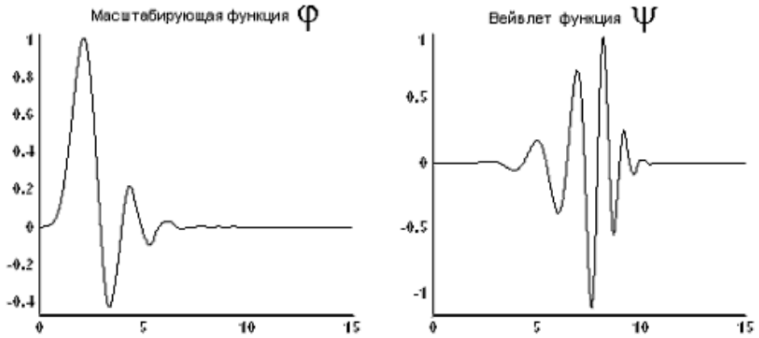

Рисунок 3 – Аналитические графики функций φ и Ψ, вейвлет Добеши 8

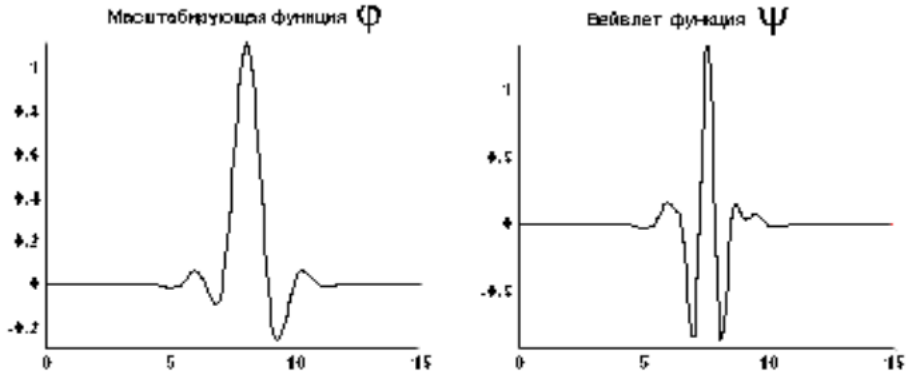

Рисунок 4 – Аналитические графики функций φ и Ψ, вейвлет Симлета 8

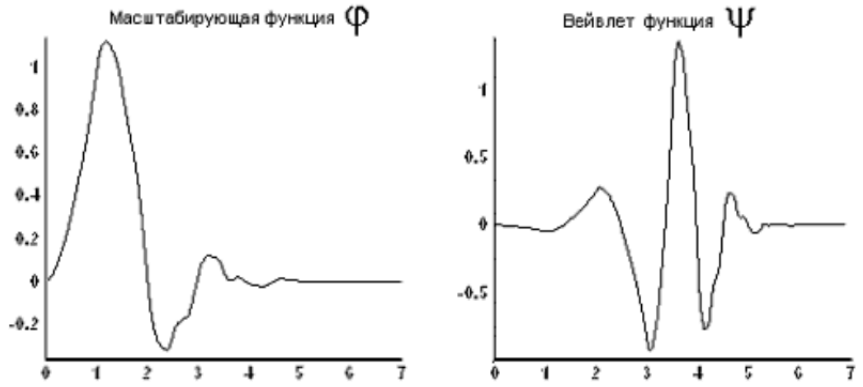

Рисунок 5 – Аналитические графики функций φ и Ψ, вейвлет Добеши 4

Масштабирование и смещение функций $\varphi_{j,k}$ и $\Psi_{j,k}$ находится следующим образом:

$$\varphi_{j,k} = 2^{j/2}\varphi(2^j x - k), \tag{7}$$

$$\Psi_{j,k} = 2^{j/2}\Psi(2^j x - k), \tag{8}$$

В свою очередь сами функции φ и Ψ определяются следующим образом:

$$\varphi = \sqrt{2}\sum_{k=0}^{2M-1} h_k \varphi(2x - k), \tag{9}$$

$$\Psi = \sqrt{2}\sum_{k=0}^{2M-1} g_k \Psi(2x - k), \tag{10}$$

где $g_k = (-1)^k h_{2M-k-1}$

Таким образом, ортогональный вейвлет-анализ сводится к нахождению коэффициентов аппроксимации и детализирующих коэффициентов в разложении сигнала $f(t)$ по формуле (5).

Достоинства вейвлет-преобразования:

- вейвлетные преобразования обладают всеми достоинствами преобразований Фурье;
- вейвлетные базисы могут быть хорошо локализованными как по частоте, так и по времени. При выделении в сигналах хорошо локализованных разномасштабных процессов можно рассматривать только те масштабные уровни разложения, которые представляют интерес;
- базисные вейвлеты могут реализоваться функциями различной гладкости.

Недостатки:

- относительная сложность преобразования.

1.2.4 Преобразование Гильберта-Хуанга

Под преобразованием Гильберта-Хуанга (Hilbert-Huang transform – HHT) понимается метод эмпирической модовой декомпозиции (EMD) нелинейных и нестационарных процессов и Гильбертов спектральный анализ (HSA) [10]. Этот метод потенциально жизнеспособен для нелинейного и нестационарного анализа данных [9], специально для частотно-энергетических временных представлений.

EMD-HSA предложил Норден Хуанг в 1995 в США (NASA) для изучения поверхностных волн тайфунов, включая возможность на анализ произвольных временных рядов коллективом соавторов в 1998 г.. В последующие годы, активно расширяя применения алгоритма для других новых отраслей науки и техники, взамен термина EMD-HSA был принят более короткий термин преобразования HHT.

EMD (Empirical Mode Decomposition) – метод разложения сигналов на функции, получившие названия внутренних или «эмпирических мод». Метод

представляет собой адаптивную итерационную вычислительную процедуру разложения исходных данных (непрерывных или дискретных сигналов) на эмпирические моды или внутренние колебания.

Огибающие сигналов. У каждого сигнала присутствуют локальные экстремумы: чередующиеся локальные максимумы и локальные минимумы с произвольным расположением по координатам (независимым переменным) сигналов. По этим экстремумам с использованием методов аппроксимации можно построить две огибающие сигналов: нижнюю – построенную по точкам локальных минимумов, и верхнюю – построенную по точкам локальных максимумов, а также функцию «среднего значения огибающих», которой отвечает срединная линия, расположенная в точности между нижней и верхней огибающими [10].

Модовая декомпозиция сигналов основана на предположении, что любые данные состоят из различных внутренних колебаний IMF. В любой момент времени данные могут иметь множество сосуществующих внутренних колебаний – IMFs. Каждое колебание, линейное или нелинейное, представляет собой модовую функцию, которая имеет экстремумы и нулевые пересечения. Кроме того, колебания в определенной степени «симметричны» относительно локального среднего значения. Конечные сложные данные образуются суммой модовых функций, наложенных на региональный тренд сигнала.

Эмпирическая мода – это такая функция, которая обладает следующими свойствами:

1) количество экстремумов функции (максимумов и минимумов) и количество пересечений нуля не должны отличаться более чем на единицу;
2) в любой точке функции среднее значение огибающих, определенных локальными максимумами и локальными минимумами, должно быть нулевым.

IMF представляет собой колебательный режим, но вместо постоянной амплитуды и частоты, как в простой гармонике, у IMF могут быть переменная амплитуда и частота, как функции независимой переменной. Первое свойство гарантирует, что локальные максимумы функции всегда положительны, локальные минимумы соответственно отрицательны, а между ними всегда имеют место пересечения нулевой линии. Второе свойство гарантирует, что мгновенные

частоты функции не будут иметь нежелательных флуктуаций, являющихся результатом асимметричной формы волны.

Любую функцию и любой произвольный сигнал, изначально содержащие произвольную последовательность локальных экстремумов (минимум 2), можно разделить на семейство функций IMFs и остаточный тренд. Если данные лишены экстремумов, но содержат точки перегиба («скрытые» экстремумы наложения модовых функций и крутых трендов), то для открытия экстремумов может использоваться дифференцирование сигнала.

Схема преобразования Гильберта-Хуанга делится на две части. На первом этапе, экспериментальные данные разлагаются в ряд внутренних модовых функций (IMFs). Эта декомпозиция рассматривается как расширение данных в терминах внутренних модовых функций. Иначе, эти внутренние модовые функции представлены как базис преобразования, которое может быть линейным или нелинейным, как диктуется по условиям. Так как IMFs имеют хорошие Гильбертовы преобразования, то могут быть вычислены соответствующие мгновенные частоты. На следующем этапе локализуются любые явления, как во времени, так и на частотной оси. Локальная энергия и мгновенная частота, выведенная из IMFs, дают дистрибутивные «энергетические время-частотные» данные, и такое представление, определяемое как Гильбертов спектр:

$$r_j(t) = r_{j-1} - c_j(t) \tag{11}$$

Пусть имеется произвольный сигнал $y(t)$. Сущность метода EMD заключается в последовательном вычислении функций эмпирических мод $c(t)$ и остатков $r_j(t) = r_{j-1} - c_j(t)$, где $j = 1, 2, 3, .., n$ при $r_0 = y(t)$. Результатом разложения будет представление сигнала в виде суммы модовых функций и конечного остатка:

$$x(t) = \sum_{j=1}^{n} c_j(t) + r_n(t), \tag{12}$$

где n - количество эмпирических мод, которое устанавливается в ходе вычислений.

Алгоритм эмпирической декомпозиции сигнала складывается из следующих операций преобразования:

1) находится в сигнале $y(k)$ положение всех локальных экстремумов, максимумов и минимумов процесса (номера точек $k_{i.ext}$ экстремумов), и значения в этих точках (рисунок 6). Между этими экстремумами сосредоточена вся информация сигнала. Группируются массивы координат $k_{i.ext}$ для максимумов и минимумов, и соответствующих им амплитудных значений $y(k_{i.ext})$. Число строк в массивах максимумов и минимумов не должно отличаться более чем на 1.

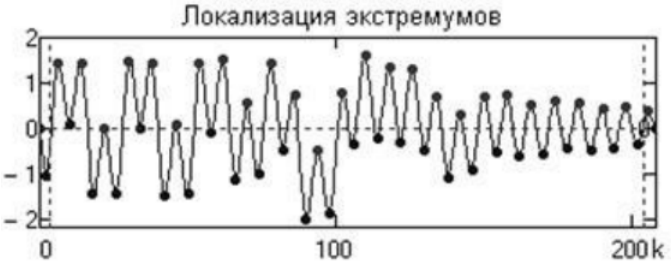

Рисунок 6 – Локализация экстремумов в сигнале

2) Применяя сплайны вычисляются верхняя $u_t(k)$ и нижняя $u_b(k)$ огибающие процесса соответственно, по максимумам и минимумам, как это показано на рисунке 7. Определяется функция средних значений $m_1(k)$ между огибающими (рисунок 7).

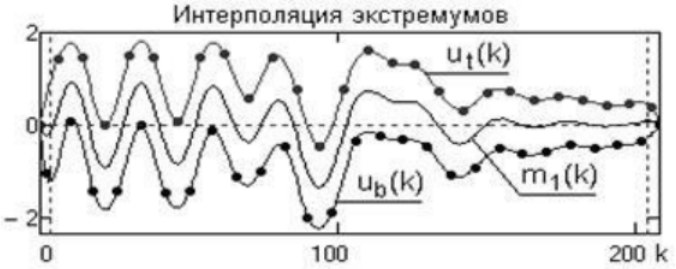

Рисунок 7 – Интерполяция экстремумов и построение огибающих

Далее, определяется функция средних значений $m_1(k)$ между огибающими.

$$m_1(k) = \frac{u_t(k) + u_b(k)}{2}, \tag{13}$$

Разность между сигналом $y(k)$ и функцией $m_1(k)$ дает первую компоненту отсеивания - функцию $h_1(k)$, которая является первым приближением к первой функции IMF:

$$h_1(k) = y(k) - m_1(k), \tag{14}$$

3) Повторяются операции 1) и 2), принимая вместо $y(k)$ функцию $h_1(k)$, и находятся второе приближение к первой функции IMF – функцию $h_2(k)$.

$$h_2(k) = h_1(k) - m_2(k), \tag{15}$$

Последующие итерации выполняются аналогичным образом:

$$h_i(k) = h_{i-1}(k) - m_i(k), \tag{16}$$

По мере увеличения количества итераций функция $m_i(k)$ стремится к нулевому значению, а функция $h_i(k)$ – к неизменяемой форме.

Последнее значение $h_i(k)$ итераций принимается за наиболее высокочастотную функцию $r_1(k) = y(k) - c_1(k)$ семейства IMF, которая непосредственно входит в состав исходного сигнала $y(k)$. Это позволяет вычесть $c_1(k)$ из состава сигнала и оставить в нем более низкочастотные составляющие $r_1(k) = y(k) - c_1(k)$,

На рисунке 8 показано графическое представление вычитания из сигнала высокочастотной составляющей, сформированной по алгоритму, заданному (12)-(16).

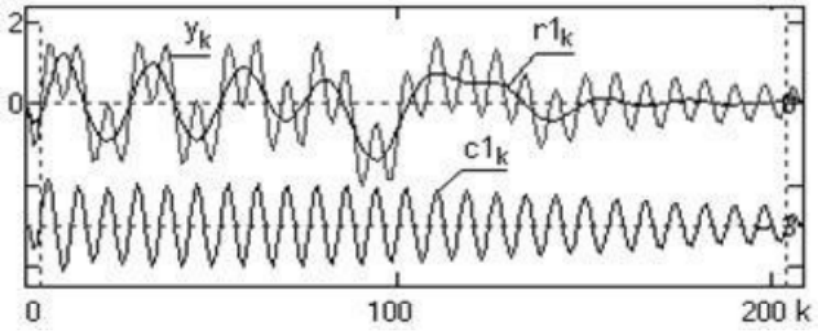

Рисунок 8 – Выявление низкочастотных составляющих в сигнале

Следующая внутренняя модовая функция находится, повторяя операции 1)-3) декомпозиции, описанные выше, с той разницей, что входным сигналом является остаток $r_1(k)$.

Шаги 1)-3) повторяются для всех последующих $r_j(k)$, и результат будет представлять последовательность вычислений:

$$r_1(k) - c_2(k) = r_2(k), ..., r_{n-1}(k) - c_n(k) = r_n(k), \tag{17}$$

Метод EMD закончен, когда остаток, в идеале, не содержит экстремумов. Это означает, что остаток – или константа или монотонная функция. Извлечённые IMFs симметричны, имеют уникальные локальные частоты, различные IMFs не показывают ту же самую частоту в то же самое время. Другими словами, остановка декомпозиции сигнала должна происходить при максимальном «выпрямлении» остатка, то есть превращения его в тренд сигнала по интервалу задания с числом экстремумов не более 2-3 [10].

Достоинства преобразования Гильберта-Хуанга:

- базис, используемый при разложении (набор эмпирических мод) конструируется непосредственно из того сигнала, с которым ведется работа, что позволяет учесть все его локальные особенности, внутреннюю структуру, наличие нежелательных особенностей (шумы, тренды, аномальные выбросы, пропущенные значения).

Недостатки:

- гораздо более высокая вычислительная сложность по сравнению с вейвлет-преобразованием.

1.3 Алгоритмы классификации

После получения акустических характеристик сигнала на заданных интервалах времени необходимо определить границы между основными лингвистическими элементами языка.

Большинство работ, связанных с автоматической сегментацией речевого сигнала, для решения данной задачи используют определенные закономерности, эвристики в изменении значений выделенных признаков речевого сигнала

в момент межфонемного перехода [13]. Эти закономерности чаще всего не имеют большой теоретической базы и лишь основываются на экспериментальном опыте других работ.

Так, в работах [7], [8] сегментация речевого сигнала на фрагменты, осуществляющая на базе Вейвлет-преобразования, используют тот факт, что на межфонемных переходах сигнал претерпевает значительные изменения сразу на многих масштабах исследования и, соответственно, характеризуется возрастанием вейвлет-коэффициентов для многих уровней детализации, в то время как на стационарных участках фонем вейвлет-коэффициенты оказываются сгруппированными вблизи определённых масштабов. Таким образом, отыскание межфонемных границ сводится к отысканию моментов увеличения вейвлет-коэффициентов на значительном количестве уровней масштабирования.

На основе этих закономерностей формируется критерий определения межфонемных переходов, который в свою очередь также определяется экспериментальным путем, в виду чего не гарантируют, полученной в результате экспериментов точности на других данных из-за сложной структуры звукового сигнала.

Исходя из приведенных выше рассуждений в данной работе решено применить метод классификации машинного обучения.

1.3.1 Формальная постановка задачи классификации

Пусть X — множество описаний объектов,

Y - конечное множество меток классов.

Существует неизвестная целевая зависимость — отображение $y^* : X \to Y$, значения которой известны только на объектах конечной обучающей выборки $X^m = \{(x_1, y_1), \ldots, (x_m, y_m)\}$.

Требуется построить алгоритм $a : X \to Y$, способный классифицировать произвольный объект $x \in X$.

Для решения этой задачи могут быть использованы следующие методы классификации [14]:

- вероятностные («наивный» Байес);
- метрические (метод k ближайших соседей);
- логические (решающее дерево);
- линейные (метод опорных векторов, логистическая регрессия);
- методы на основе искусственных нейронных сетей (FFBP, RNN, CNN).

Рассмотрим наиболее популярные подходы для каждого из методов.

1.3.2 Наивный байесовский классификатор

Метод Байеса (Naive Bayes, NB) относится к вероятностным методам классификации.

Пусть $P(c_i|d)$ - вероятность того, что объект, представленный вектором $d = (t_1, ..., t_n)$, соответствует категории c_j для $i = 1, ...|C|$. Задача классификатора заключается в том, чтобы подобрать такие значения c_i и d, при которых значение вероятности $P(c_i|d)$ будет максимальным:

$$CSV(d) = \underset{c_i \in C}{argmax} P(c_i|d) \tag{18}$$

Для вычисления значений $P(c_i|d)$ используется теорема Байеса:

$$P(c_i|d) = \frac{P(c_i)P(d|c_i)}{P(d)}, \tag{19}$$

где

- $P(c_i)$ - априорная вероятность того, что вектор d отнесен к категории c_i;
- $P(d|c_i)$ - вероятность найти вектор $d = (t_1, ..., t_n)$ в категории c_i;
- $P(d)$ - вероятность того, что произвольно взятый объект можно представить в виде вектора признаков $d = (t_1, ..., t_n)$.

$P(d)$ не зависит от категории c_i, а значения $t_1, ..., t_n$ заданы заранее, поэтому знаменатель не влияет на выбор наибольшего из значений $P(c_i|d)$.

Вычисление $P(d|c_i)$ затруднительно из-за большого количества признаков $t_1, ..., t_n$ поэтому делают «наивное» предположение о том, что любые случайные величины не зависят друг от друга.

Исходя из этого допущения можно воспользоваться формулой:

$$P(d|c_i) = \prod_{k=1}^{n} P(t_k|c_i) \tag{20}$$

Преимущества метода:

- высокая скорость работы;
- поддержка инкрементного обучения;
- относительно простая реализация программная реализация алгоритма;

- легкая интерпретируемость результатов алгоритма.

Недостатки метода:

- относительно низкое качество классификации по сравнению с другими методами;
- неспособность учитывать зависимость результата классификации от сочетания признаков из-за допущения о их независимости друг от друга.

1.3.3 Метод k-ближайших соседей

Метод k ближайших соседей (k Nearest Neighbours, KNN) относится к метрическим методам классификации.

Чтобы найти категорию, соответствующую объекту d, классификатор сравнивает d со всеми объектами из обучающей выборки L, то есть для каждого $d_z \in L$ вычисляется расстояние (d_z, d). Далее из обучающей выборки выбираются k объектов, ближайших к d. Согласно методу k ближайших соседей, объект d считается принадлежащим тому классу, который является наиболее распространенным среди соседей данного объекта, то есть для каждого класса c_i вычисляется функция ранжирования:

$$CSV(d) = \sum_{d_z \in L_k(d)} p(d_z, d) * \Phi(d_z, c_i), \tag{21}$$

где

- $L_k(d)$ - ближайшие k объектов из L к d;
- $\Phi(d_z, c_i)$ - количество объектов d_z обучающей выборки относящиеся к категории c_i.

Преимущества метода:

- возможность обновления обучающей выборки без переобучения классификатора;
- устойчивость алгоритма к аномальным выбросам в исходных данных;
- относительно простая программная реализация алгоритма;
- легкая интерпретируемость результатов работы алгоритма;
- хорошее обучение в случае с линейно неразделимыми выборками.

Недостатки метода:

- репрезентативность набора данных, используемого для алгоритма;
- высокая зависимость результатов классификации от выбранной метрики;
- большая длительность работы из-за необходимости полного перебора обучающей выборки;
- невозможность решения задач большой размерности по количеству классов и объектов.

1.3.4 Метод деревьев решений (Decision Trees, DT)

Метод деревьев решений (Decision Trees, DT) относится к логическим методам классификации.

Деревом решений называют ациклический граф, по которому производится классификация объектов, описанных набором признаков. Каждый узел дерева содержит условие ветвления по одному из признаков. У каждого узла столько ветвлений, сколько значений имеет выбранный признак. В процессе классификации осуществляются последовательные переходы от одного узла к другому в соответствии со значениями признаков объекта.

Классификация считается завершенной, когда достигнут один из листьев (конечных узлов) дерева. Значение этого листа определит класс, которому принадлежит рассматриваемый объект. На практике обычно используют бинарные деревья решений, в которых принятие решения перехода по ребрам осуществляется простой проверкой наличия признака в объекте. Если значение признака меньше определенного значения, выбирается одна ветвь, если больше или равно, другая.

Преимущества метода:

- относительно простая программная реализация алгоритма;
- легкая интерпретируемость результатов работы алгоритма.

Недостатки метода:

- неустойчивость алгоритма к выбросам исходных данных;
- требуется большой объем данных для получения точных результатов.

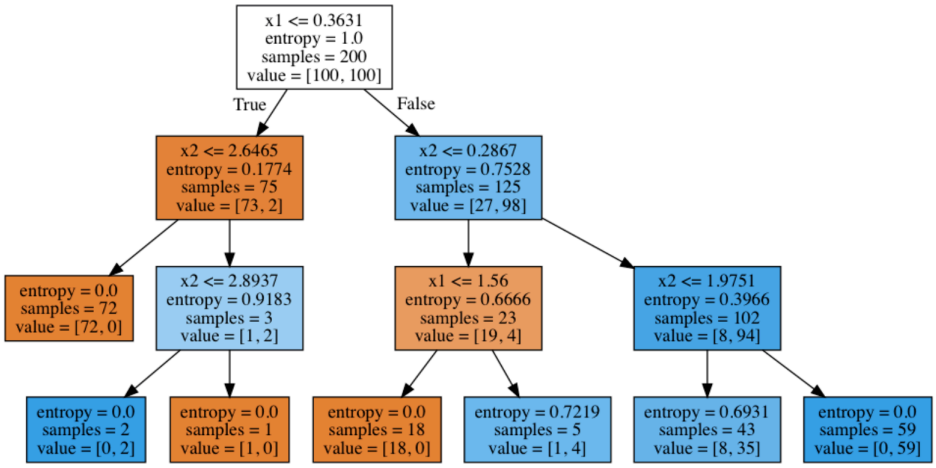

Рисунок 9 – Пример построенного бинарного дерева решений по двум признакам

1.3.5 Метод опорных векторов

Метод опорных векторов (Support Vector Machine, SVM) является линейным методом классификации. Рассмотрим множество объектов, которые необходимо классифицировать. Сопоставим ему множество точек в пространстве размерности |D|.

Выборку точек называют линейно разделимой, если принадлежащие разным классам точки можно разделить с помощью гиперплоскости (в двумерном случае - прямой). Необходимо провести прямую так, чтобы по одну сторону от нее лежали все точки одного класса, а по другую - все точки другого класса. Тогда для классификации неизвестных точек достаточно будет посмотреть, с какой стороны прямой они окажутся.

В методе опорных векторов расстоянием между прямой и множеством точек считается расстояние между прямой и ближайшей к ней точкой из множества. Именно такое расстояние и максимизируется в данном методе. Гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей, называется разделяющей (на рисунке 10 обозначена буквой L). Ближайшие к параллельным гиперплоскостям точки называются опорными векторами, через них проходят пунктирные линии.

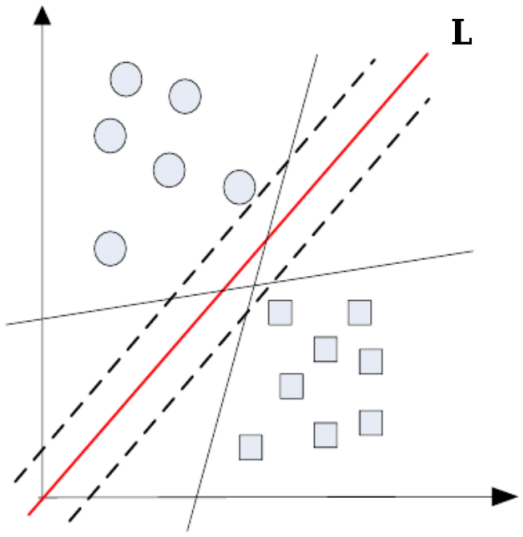

Рисунок 10 – Разделяющая гиперплоскость в методе опорных векторов

Преимущества метода:

- один из наиболее качественных методов;
- возможность работы с небольшим набором данных для обучения;
- сводимость к задаче выпуклой оптимизации, имеющей единственное решение.

Недостатки метода:

- сложная интерпретируемость параметров алгоритма;
- неустойчивость к выбросам в исходных данных.

1.3.6 Методы на основе искусственных нейронных сетей

Нейронная сеть — это система соединённых и взаимодействующих между собой простых процессоров (нейронов), соединенных между собой синапсами.

Существует большое количество разновидностей нейронных сетей, основные из них – сети прямого распространения, рекуррентные сети, радиально-базисные функции и самоорганизующиеся карты.

Рассмотрим структуру классической нейронной сети прямого распространения (Feed Forward Back Propagation, FFBP).

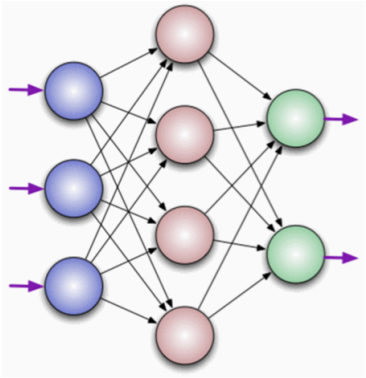

Рисунок 11 – Пример нейронных слоев (синий - входной, красный - скрытый, зеленый - выходной)

В нейронных сетях прямого распространения присутствуют входной слой, выходной слой и промежуточные (скрытые) слои: сигнал идет последовательно от входного слоя нейронов по промежуточным слоям к выходному.

Для классификации объекта d_i при помощи нейронной сети прямого распространения веса признаков документа подаются на соответствующие входы сети. Активация распространяется по сети; значения, получившиеся на выходах, есть результат классификации. Стандартный метод обучения такой сети - метод обратного распространения ошибки.

Преимущества метода:

- имеет очень высокое качество алгоритма при удачном подборе параметров;
- является универсальным аппроксиматором непрерывных функций;
- поддерживает инкрементное обучение.

Недостатки метода:

- вероятность возможной расходимости или медленной сходимости, поскольку для настройки сети используются градиентные методы;
- необходимость очень большого объема данных для обучения, чтобы достичь высокой точности;
- низкая скорость обучения;
- сложная интерпретируемость параметров алгоритма.

1.4 Постановка задачи

В результате проведения анализа предметной области, существующих методов выделения признаков речевого сигнала и алгоритмов классификации, постановка задачи, которую необходимо решить в рамках данной работы, может быть определена следующим образом.

Требуется спроектировать метод выделения звуков естественного языка в звучащей речи, а также разработать программный модуль для проверки работоспособности метода.

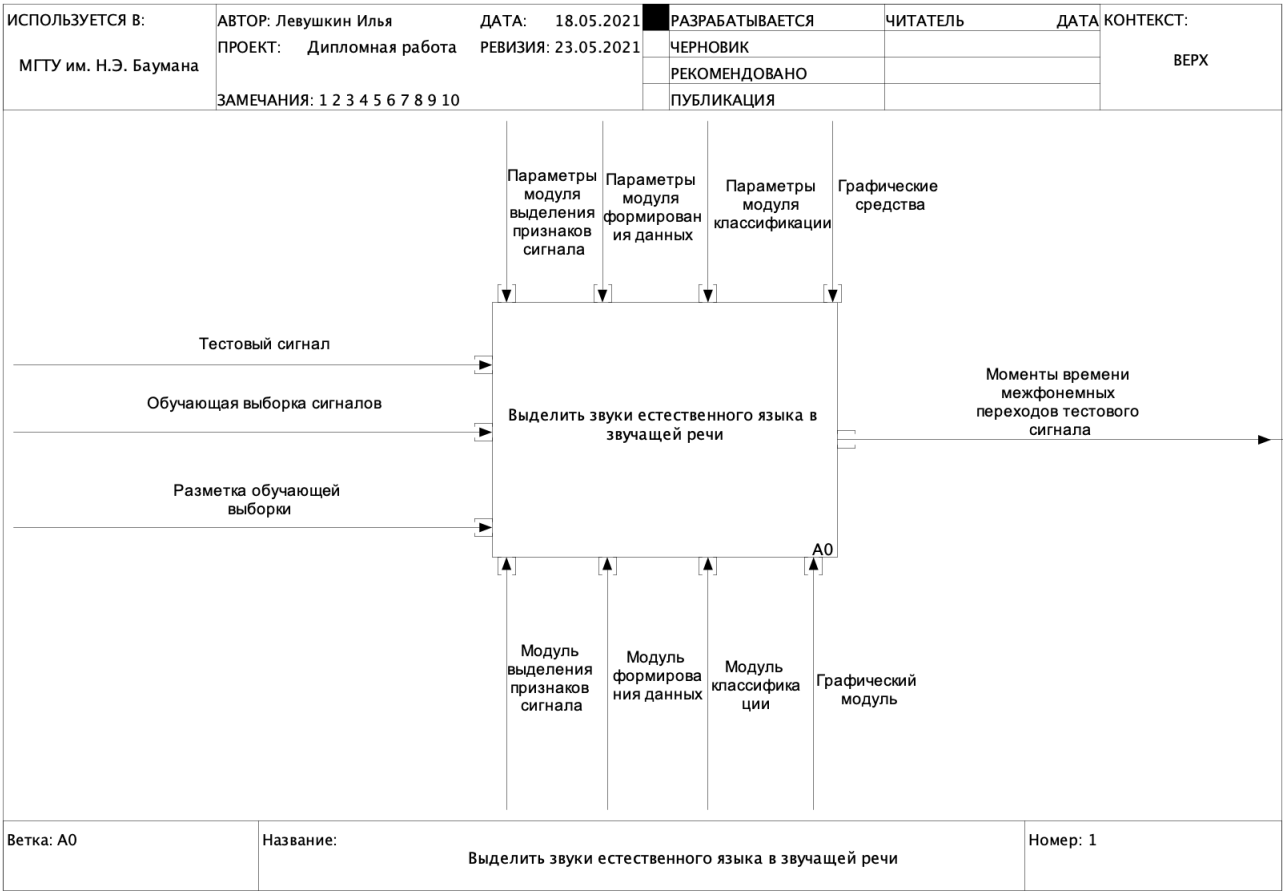

Рисунок 12 – IDEF0 диаграмма разрабатываемого модуля.

Входными данными являются аудиофайл и обучающая выборка аудиофайлов формата mp3, содержащие в себе информацию об амплитуде и частоте звуков в дискретные моменты времени, а также разметка обучающей выборки аудиофайлов в csv формате.

Выходным параметром является список моментов времени, на которых сигнал содержит межфонемные переходы.

Спроектированный метод должен обеспечить возможность

- выбрать тестовый аудиофайл;
- вычислить список моментов времени, на которых сигнал содержит межфонемные переходы и визуализировать их графически на осциллограмме (амплитудно-временном графике сигнала) [16].

1.5 Выводы

В ходе анализа предметной области были описаны основные этапы распознавания речевого сигнала, а также было дано обоснование актуальности задачи сегментации речевого сигнала.

Был проведен анализ существующих решений сегментации речевого сигнала на фрагменты, на основе которых были выделены два основных этапа в задаче сегментации:

- выделение признаков сигнала;
- определение межфонемных переходов на основе выделенных признаков сигнала.

Был произведен анализ существующих методов выделения признаков речевого сигнала. Ниже приведена сводная таблица особенностей каждого из рассмотренных методов относительно друг друга.

Метод	Базис	Представление	Нелинейность	Нестационарность	Вычислительная сложность
Фурье	Априорный	Энергия-частота	Нет	Нет	Низкая
Вейвлет	Априорный	Энергия-время-частота	Нет	Да	Средняя
Гильберт-Хуанг	Адаптивный	Энергия-время-частота	Да	Да	Высокая

Таблица 1 – Сравнение существующих методов выделения признаков речевого сигнала.

В данной работе было решено использовать метод, основанный на Вейвлет преобразовании, а именно, многомасштабный Вейвлет-анализ, поскольку он учитывает нестационарность человеческой речи, а также имеет относительно невысокую вычислительную сложность по сравнению с преобразованием Гильберта-Хуанга, что имеет большое значение в системах распознавания речи.

Также, было обнаружено, что большинство работ используют закономерности, эвристики, основанные на экспериментальном опыте других работ и не имеющие под собой существенной теоретической базы, в виду чего эти решения не гарантируют, полученной в результате экспериментов, точности на других данных.

Поэтому, в качестве альтернативы было решено использовать наиболее оптимальный для задачи определения межфонемных интервалов сигнала алгоритм классификации на основе выделенных признаков сигнала.

Ниже приведена сводная таблица наиболее важных особенностей для решения поставленной задачи каждого из методов классификации относительно друг друга для малого объема обучающей выборки.

Метод	Качество обучения	Скорость работы	Поддержка инкрементного обучения
Наивный Байес	Низкое	Высокая	Есть
k-ближайших соседей (KNN)	Среднее	Низкая	Есть
Деревья решений (DT)	Среднее	Средняя	Нет
Опорные вектора (SVM)	Высокое	Средняя	Нет
Искусственные нейронные сети	Среднее	Низкая	Есть

Таблица 2 – Сравнение существующих методов классификации объектов.

В данной работе было решено использовать метод опорных векторов из-за недостаточно большого объема обучающей выборки. Метод SVM дает наилучшее качество обучения на небольшом наборе данных, а также не требует значительных ресурсных затрат для обучения и предсказания результатов по сравнению с другими методами.

2 Конструкторская часть

Данный раздел содержит в себе следующие задачи:

- изложение принципов спроектированного метода выделения звуков естественного языка в звучащей речи;
- подробное описание всех этапов разработанного алгоритма, реализующего данный метод;
- проектирование архитектуры системы, проверяющей работоспособность данного метода.
- описание формата входных и выходных данных и структуры ПО;

2.1 Метод выделения звуков естественного языка в звучащей речи

Данный метод состоит из двух основных этапов:

- формирования признаков из поступающего на вход речевого сигнала одним из существующих методов выделения признаков сигнала;
- получения границ межфонемных переходов из сформированных признаков сигнала при помощи применения алгоритмов классификации.

На вход данный метод получает цифровой сигнал, содержащий в себе значения амплитуд (уровней сигнала) в дискретные моменты времени, а на выходе выдает список границ (моментов времени в секундах) межфонемных переходов (переходов с одного лингвистического элемента языка [1] к другому) естественного языка в звучащей речи.

2.2 Описание алгоритма метода выделения звуков естественного языка в звучащей речи

Алгоритм, реализующий данный метод, состоит из следующих этапов:

1) выделение признаков речевых сигналов из проверяемого сигнала и обучающей выборки (рисунок 13 блок А1);
2) формирование данных для применения на них (обучения и предсказания результатов) модели классификации (рисунок 13 блок А2);
3) обучение модели классификации на сформированной обучающей выборке (рисунок 13 блок А3);
4) применение модели классификации на тестовых данных (проверяемом сигнале) для получения предсказанных значений - межфонемных переходов (рисунок 13 блок А4);

5) графическое представление результатов предсказания (рисунок 13 блок A5).

Ниже приведена IDEF0-диаграмма иллюстрирующая основные этапы алгоритма.

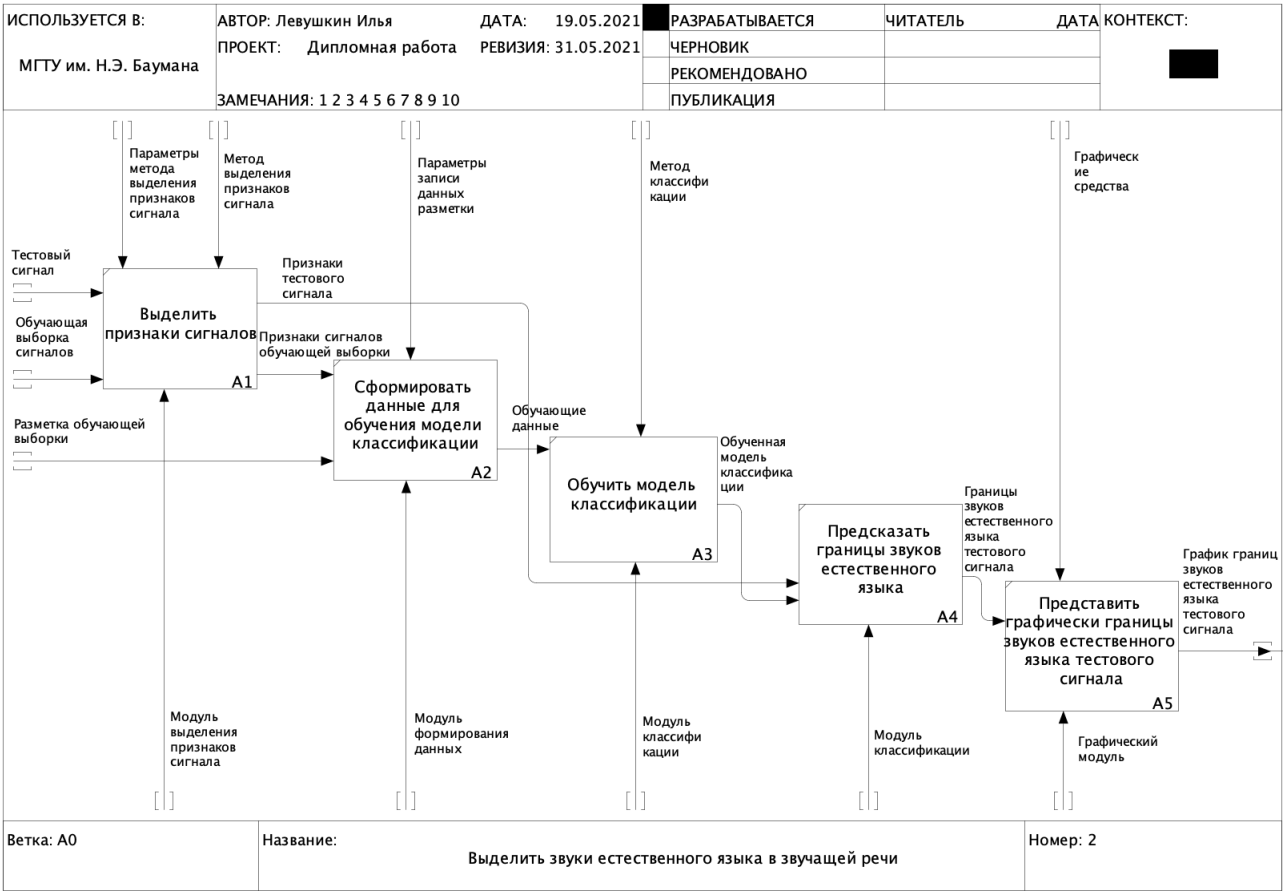

Рисунок 13 – IDEF0 диаграмма разрабатываемого алгоритма.

2.3 Выделение признаков речевого сигнала

В данной работе было решено использовать метод многомасштабного Вейвлет-анализа сигнала (раздел 1.5), в результате чего схема алгоритма выделения признаков речевого сигнала выглядит следующим образом [7]:

1) В качестве предобработки сигнал нормализуется: все отсчеты делятся на максимальное значение, для установки единых пороговых значений для любых входных сигналов;
2) Входной сигнал разбивается на фреймы по n_{frame} отсчётов при частоте дискретизации sr с перекрытием hop.
3) Каждый фрейм накрывается окном Хэмминга для устранения дефектов на краях;

4) К каждому фрейму применяется вейвлет-преобразование в базисе $base$. Используется разложение до k-го уровня декомпозиции;
5) Для каждого уровня декомпозиции определяется энергия, как сумма квадратов значений коэффициентов детализации E [6].

Ниже приведены варьируемые параметры, поступающие на вход модулю выделения принаков сигнала:

- n_{frame} - количество отсчетов в одном фрейме (по умолчанию - 2500);
- sr - частота дискретизации сигнала, Гц (по умолчанию - 16000);
- hop - размер перекрытия фрейма другим фреймом, % (по умолчанию - 40);
- $base$ - базисная функция вейвлет-пребразования (по умолчанию - Вейвлет Добеши 8);
- k - уровень декомпозиции сигнала (по умолчанию - 6).

2.4 Формирование данных для обучения модели классификации

Из формализованной постановки задачи классификации (раздел 1.3.1) следует, что для обучения модели классификации необходимо сформировать обучающую выборку таким образом, чтобы каждому вектору признаков $(x^1, x^2, ..., x^d)$ объекта x множества X^m соответствовало единственное значение ответа $y = y(x)$ множества Y^m:

$$X^m = (x_1, y_1), ..., (x_m, y_m). \tag{22}$$

В терминах данной задачи это означает, что каждому вектору признаков речевого сигнала за определенный промежуток времени t требуется сопоставить ответ на вопрос - *является ли данный интервал t границей межфонемного перехода?*

Говоря формально, требуется сформировать список ответов Y^m, элементы y_i которого принимают значения $\{0, 1\}$:

- 1 - в случае если объект x_i (вектор признаков речевого сигнала на интервале времени t_i) является границей межфонемного перехода;
- 0 - в противном случае.

Исходя из приведенных выше рассуждений, схема формирования данных для модели классификации выглядит следующим образом:

1) Извлечение границ (моментов времени) $T = (t_1, ..., t_s)$ межфонемных переходов из файла с обучающей выборкой в соответствии с ее параметрами

записи данных;

2) Избавление от зашумленных данных путем усреднения значений t_i и t_{i+1} при условии, что $t_{i+1} - t_i < lim_{phonem}$, где lim_{phonem} - минимальный порог длительности фонемы (если расстояние между соседними значениями t меньше порогового, то заменяем их средним значением);
3) Преобразование границ межфонемных переходов $T = (t_1, ..., t_s)$ в список ответов $Y = (y_1, ..., y_m)$, где $y_i \in 0, 1$.

2.5 Обучение модели классификации

В данной работе было решено использовать метод опорных векторов SVM (раздел 1.5) в качестве алгоритма классификации.

Ниже приведены настраиваемые параметры метода:

- kernel - ядро SVM, задает функцию разделяющей гиперплоскости. Наиболее популярные ядра - linear (линейная функция), rbf (радиально-базисная функция), poly (полиномиальная функция). В данной реализации метода выбрано по умолчанию ядро rbf;
- C - параметр регуляризации. Варьирование этого параметра позволяет понизить уровень переобучения модели, поскольку он контролирует соотношение между гладкой границей и корректной классификацией рассматриваемых точек. По умолчанию - 1.0;
- gamma - «ширина» ядра. Варьирование этого параметра также позволяет понизить уровень переобучения модели для нелинейных ядер. По умолчанию равен $\frac{1}{n_{features} * X_{var}}$;
- class_weight - параметр, позволяющий варьировать размер штрафа при неудачном выборе класса. Используется при несбалансированной выборке классов. По умолчанию равен balanced, то есть устанавливает штраф за ошибку выбора класса пропорционально размеру этого класса в обучающей выборке. Применяется, поскольку в данной задаче классы имеют значительный дисбаланс (интервалов времени, на которых присутствуют границы межфонемных переходов, значительно меньше чем тех, на которых их нет).

2.6 Структура ПО

Архитектура ПО использует внутри себя принципы архитектурного патерна MVC (Model-View-Controller), который разделяет общую структуру кода

на три отдельных компонента:

- модель - предоставление данных конкретным элементам системы;
- представление - реагирует на изменение данных в системе и обеспечивает их отображение пользователю;
- контроллер - точка входа для доступа к модели и представлению. Контроллер обрабатывает действия пользователя, после чего отдает сигнал модели о необходимости каким-либо образом измениться.

При использовании MVC достигается возможность разделения компонентов работы с данными, пользовательским интерфейсом и логикой взаимодействия пользователя с приложением, в результате чего модификация одного из компонентов оказывает минимальное воздействие на остальные.

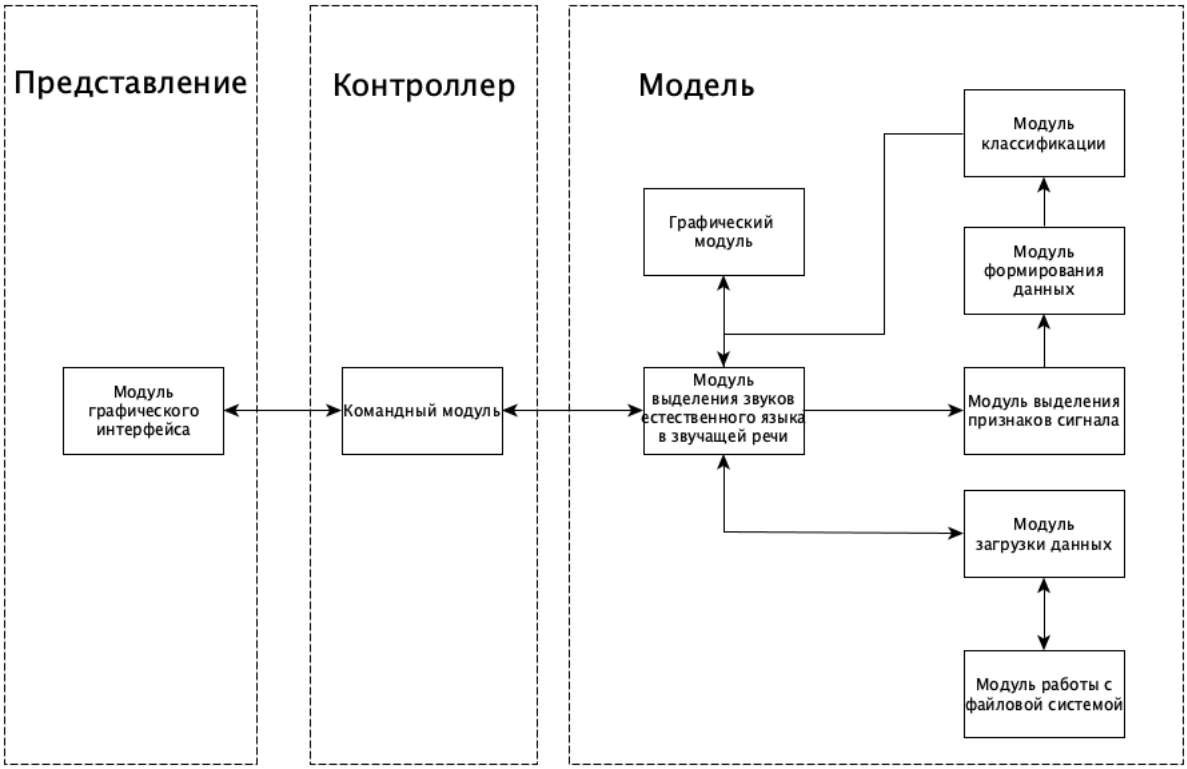

Рисунок 14 – Архитектура программного обеспечения.

Графический интерфейс предоставляет пользователю возможность взаимодействия с разработанным ПО.

Доступный функционал:

- возможность выбрать аудио-файл (формат mp3) для тестирования метода, аудио-файлы (формат mp3) и разметку (формат csv) для обучения модели

классификации;

- возможность обучить модель классификации;
- возможность сохранить обученную модель классификации в файл (формат pkl);
- возможность загрузить обученную модель классификации из файла (формат pkl);
- возможность определения границ межфонемных переходов выбранного сигнала для тестирования;
- сохранение результатов в файл (формат csv);
- графическое отображение результатов в диалоговом окне.

Модули, изображенные на рисунке 14, содержат необходимые классы и методы, выполняющие основную логику приложения.

Ниже приведено описание основных классов, представленных на рисунке 15.

- Классы *MainView, ResultView, TeachModelView* - обрабатывают логику запросов пользователя к графическому интерфейсу приложения;
- Класс *MainController* агрегирует в себе классы *MainView, ResultController, TeachModelController, SoundsExtractionModule*. Является точкой входа в приложение и отвечает за распределение задач от *MainView* к *SoundsExtractionModule* и по своим контроллерам - *ResultController*, *TeachModelController*;
- *ResultController и TeachModelController* отвечают за распределение задач от *ResultView и TeachModelView к SoundsExtractionModule*;
- *SoundsExtractionModule* агрегирует в себе все модули: *GraphModule*, *FeatureExtractionModule, DataPreprocessingModule, ClassificationModule и LoadDataModule*. Отвечает за распределение задач от контроллеров по модулям;
- *LoadDataModule* отвечает за работу с файловой системой. Осуществляет загрузку и выгрузку данных: *test_signal, train_signals, train_markups, model*;
- *ClassificationModule* отвечает за обучение модели и осуществление ею предсказаний;
- *DataPreprocessingModule* отвечает за предобратку данных. Формирует обучающую и тестовую выборки для *ClassificationModule*;

• *FeatureExtractionModule* отвечает за извлечение признаков из сигналов.

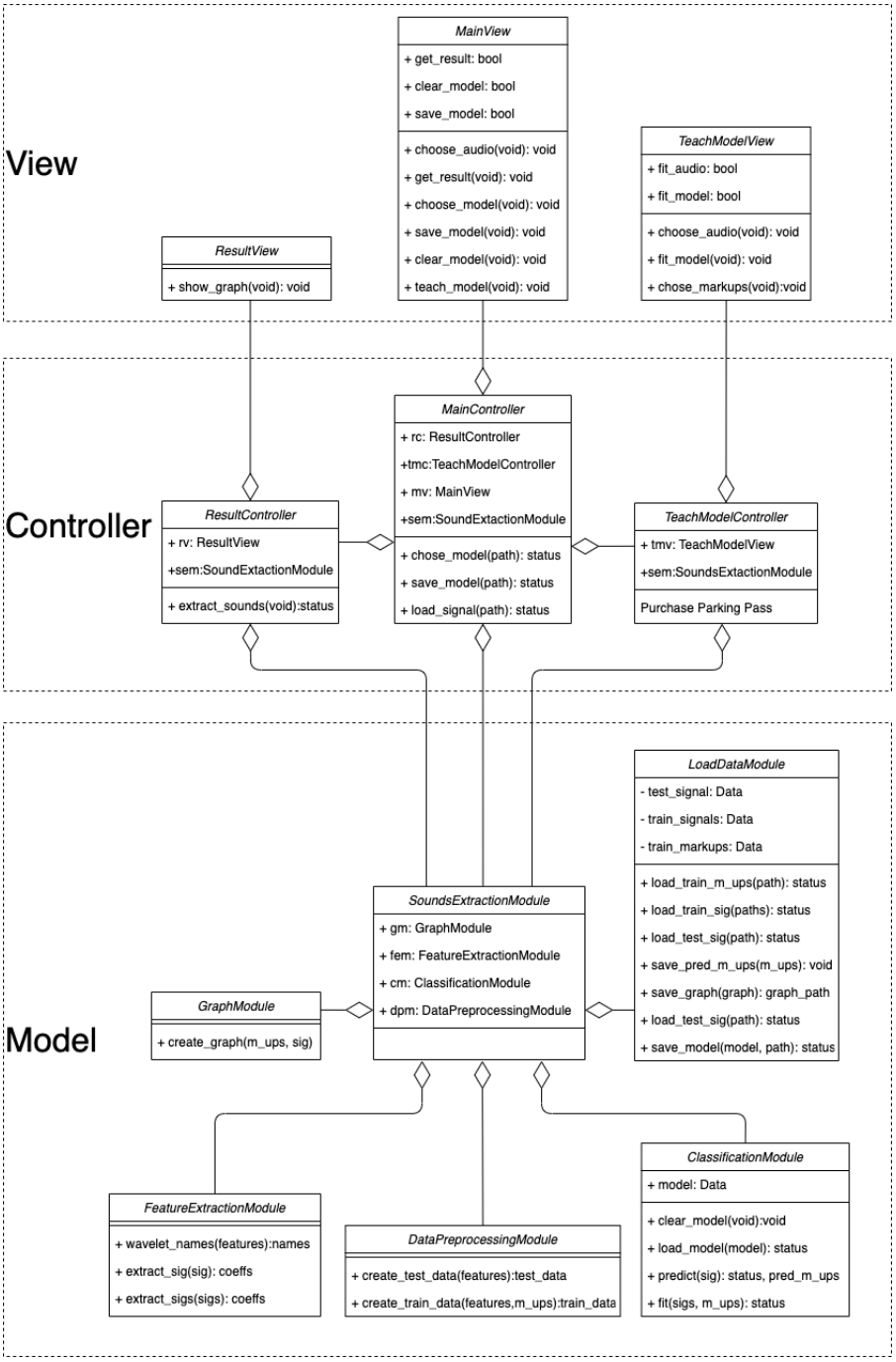

Рисунок 15 – Схематичное представление архитектуры программного обеспечения.

2.7 Описание формата входных и выходных данных

Для обучения модели пользователь выбирает файлы аудио-сигналов (формат mp3) и разметку этих аудио-файлов (формат csv). Разметка аудио-файлов представляет из себя таблицу со столбцами *filename, start, end, intervals*, где

• filename - имя аудио-файла;

- start - начало фонемы;
- end - конец фонемы;
- intervals - список границ, на которых был размечен аудио-файл.

Вместо обучения пользователь может загрузить файл обученной модели (формат pkl).

Для тестирования модели пользователь выбирает файл с аудио-сигналом (формат mp3).

На выходе пользователь получает график границ межфонемных переходов, выведенный на экран и сохраненный в файл (формат png), список выделенных границ межфонемных переходов, сохраненных в файл (формат csv).

Также, пользователь имеет возможность сохранить обученную модель в файл (в формате pkl).

2.8 Выводы

В результате написания конструкторского раздела были решены следующие задачи:

- спроектирован метод выделения звуков естественного языка в звучащей речи;
- разработан алгоритм, реализующий данный метод, и приведено подробное его описание;
- спроектирована архитектура системы для проверки работоспособности метода;
- описаны формат входных и выходных данных и структура разрабатываемого программного обеспечения.

3 Технологическая часть

3.1 Выбор языка программирования

В качестве языка программирования был выбран язык программирования *Python3* - объектно-ориентированный интерпретируемый язык программирования высокого уровня [17].

Преимуществами данного языка являются:

- встроенные типы данных высокого уровня: списки, словари, строки и другие;
- объектно-ориентированность - язык предоставляет удобный интерфейс создания и использования объектов;
- широко используется в научных исследованиях, так как обладает большим объемом различных библиотек для решения прикладных задач из различных областей, включая такие области как цифровая обработка сигналов, машинное обучение, работа со сложными математическими вычислениями и большим объемом данных.

3.2 Выбор используемых библиотек

Спроектированный метод выделения звуков естественного языка звучащей речи состоит из двух этапов (раздел 1.1.2):

1) выделение признаков сигнала;
2) определение межфонемных переходов на основе выделенных признаков сигнала.

Первый этап подразумевает работу с аудио-сигналами, а именно - загрузка аудио-файлов, дискретизация сигнала, разделение сигнала на перекрывающиеся участки - фреймы.

Для решения этих задач было решено использовать библиотеку *Librosa* [19] - модуль python, предоставляющий удобный интерфейс для анализа звуковых сигналов.

В качестве модуля выделения признаков сигнала был выбран *Многомасштабный Вейвлет-анализ* (раздел 1.5). Для решения этой задачи была выбрана библиотека *pywt* [20] - модуль python, предоставляющий необходимый инструментарий для применения вейвлет-преобразования на дискретном сигнале.

На **втором этапе** для определения границ между основными лингвисти-

ческими элементами языка было решено применить метод классификации машинного обучения (раздел 1.3.1) - метод опорных векторов SVM [15]. Наиболее популярным подходом к решению задач классификации является использование библиотеки *Scikit-Learn* [21] в связке с такими инструментами как *Pandas* [22], *NumPy* [24] и *Matplotlib* [23]. Такой подход позволяет наиболее эффективно использовать объем доступной памяти и процессорное время, предоставляемое системой.

- *Pandas* - библиотека python для обработки и анализа больших данных. Использует объекты *NumPy* и предоставляет специальные структуры данных и операции для манипулирования числовыми таблицами и временными рядами;
- *NumPy* - библиотека python, добавляющая поддержку больших многомерных массивов и матриц, вместе с большой библиотекой высокоуровневых и быстрых математических функций для операций с этими массивами;
- *Matplotlib* - библиотека python, использующаяся для построения графиков и диаграмм. Использует объекты *NumPy* и предоставляет объектно-ориентированный интерфейс для данных, отображая получившиеся результаты с помощью встроенных графических библиотек.

Графический интерфейс, используемый пользователем был написан при помощи библиотеки *PyQt5* [25] - графической библиотеки python, предоставляющей возможность создавать графические интерфейсы для пользователя. Данная библиотека использует понятную структуру наследования и предлагает объектно-ориентированные решения с логической иерархией между объектами.

3.3 Выбор среды разработки

В качестве среды разработки было решено использовать *PyCharm* [18], имеющий доступный интерфейс, а также большое количество полезных функций и плагинов, упрощающих процесс разработки.

Поскольку *PyCharm* нативно поддерживает Python, данная среда разработки выполняет большое количество рутинных действий за разработчика. Среда снабжена отладчиком, необходимым для разработки ПО. Также, среда предоставляет возможность удобного рефакторинга кода, меняя все зависимости во всем проекте.

3.4 Пользовательский интерфейс

Пользователь взаимодействует с графическим интерфейсом ПО курсором мыши и клавиатурой.

Ниже приведено описание взаимодействия пользователя с графическим интерфейсом ПО.

- **Обучение модели.**

  Для обучения модели классификации, пользователю необходимо нажать на кнопку *Обучить модель классификации*.

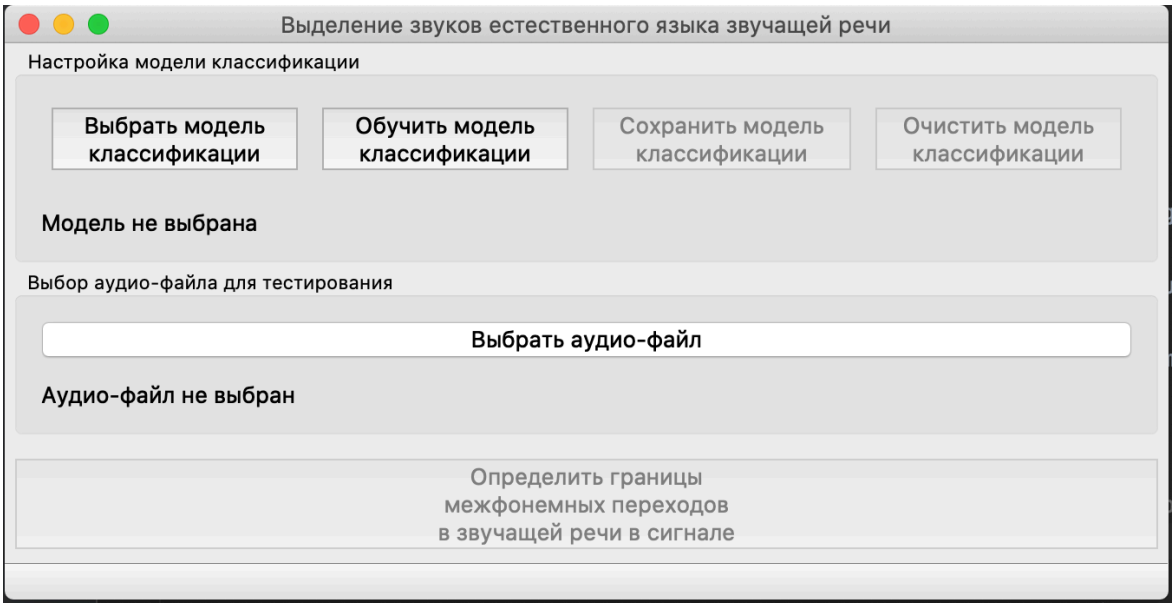

Рисунок 16 – Главное меню.

В открывшемся диалоговом окне необходимо выбрать файл-разметки аудио-сигналов для обучения модели, нажав на кнопку *Выбрать разметку для обучения*. Затем выбрать аудио-файлы для обучения модели, нажав на кнопку *Выбрать аудио-файлы для обучения*. После загрузки файлов, нажать кнопку *Обучить модель*, дождаться конца обучения и нажать кнопку *ОК*.

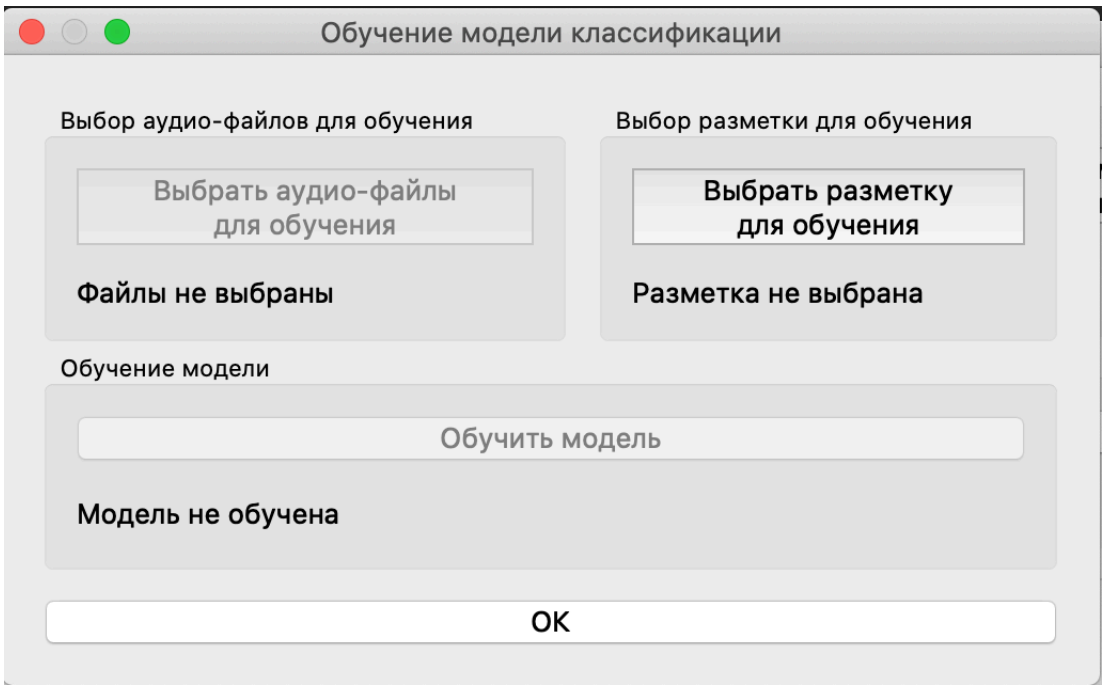

Рисунок 17 – Обучение модели.

- **Сохранение модели.**

  После обучения модели появляется возможность сохранить модель. Для этого необходимо нажать на кнопку *Сохранить модель классификации*.

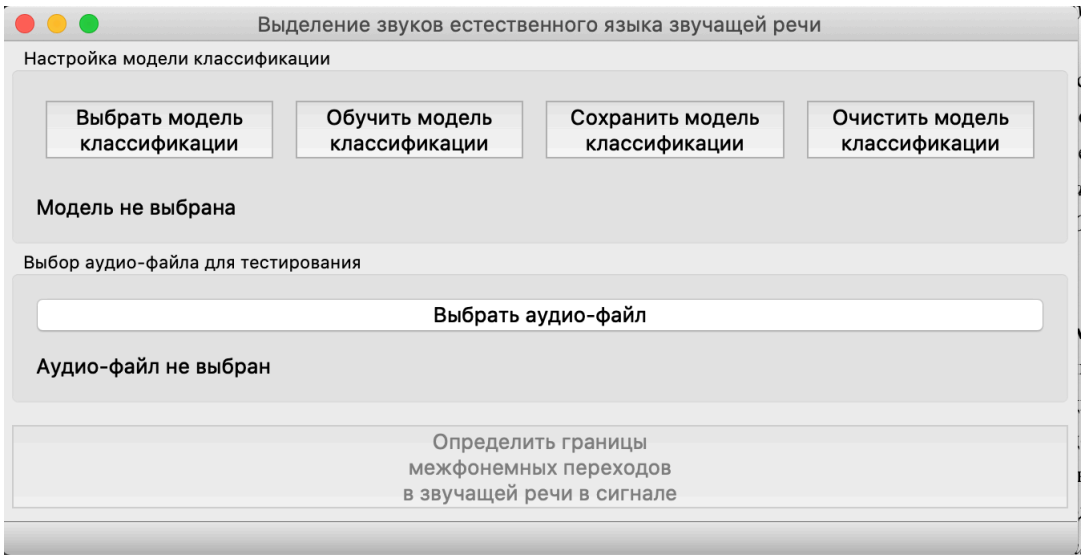

Рисунок 18 – Сохранение модели.

- **Загрузка модели.**

  Вместо обучения модели пользователь имеет возможность загрузить файл обученной модели. Для этого необходимо нажать на кнопку *Выбрать модель классификации*.

- **Сброс модели.**

  После загрузки или обучения модели классификации у пользователя появляется возможность сбросить модель. Для этого необходимо нажать на кнопку *Очистить модель классификации*.
- **Выбор аудио-файла.**

  Пользователь имеет возможность загрузить аудио-файл для тестирования ПО. Для этого необходимо нажать на кнопку *Выбрать аудио-файл*.
- **Выделение звуков естественного языка звучащей речи**

  Для получения результатов предсказания модели пользователь должен нажать на кнопку **Определить границы межфонемных переходов звучащей речи в сигнале** после выбора тестового аудио-файла и обучения/загрузки модели.

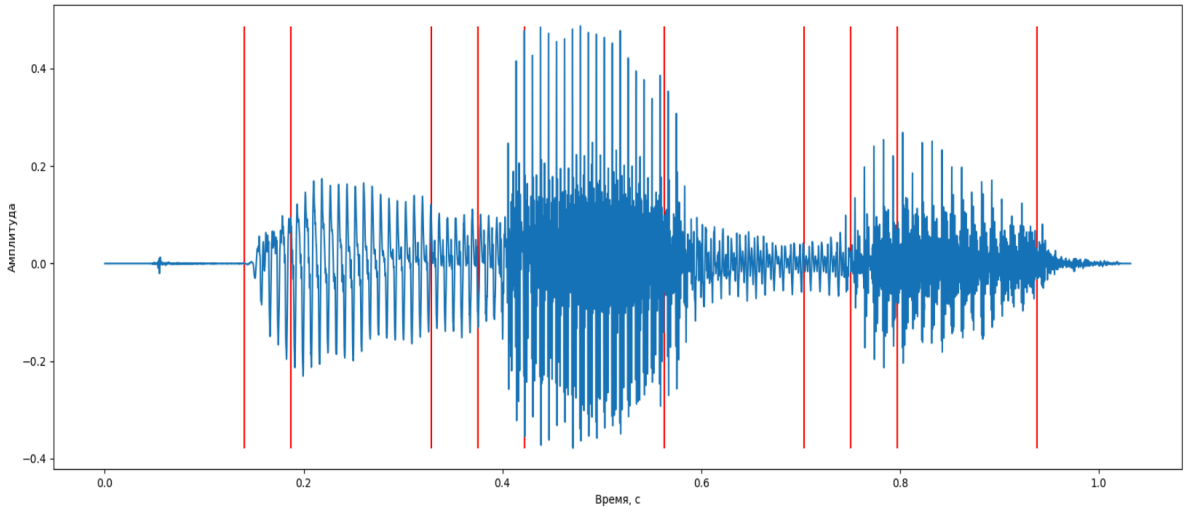

Рисунок 19 – Выделение границ межфонемных переходов звучащей речи в сигнале.

3.5 Реализация

Ниже приведены листинги кода реализаций некоторых компонентов ПО: блоков работы с данными - Model, блока работы с пользовательским интерфейсом - View, блока логики взаимодействия пользователя с приложением - Controller (полный листинг кода разработанного ПО в разделе *ПРИЛОЖЕНИЕ А*).

MainView - логика запросов пользователя к графическому интерфейсу главного экрана приложения.

Листинг 1 – MainView

```
from PyQt5 import uic
from PyQt5.QtCore import pyqtSlot, Qt
from PyQt5.QtWidgets import QMessageBox, QMainWindow, QFileDialog


class MainWindow(QMainWindow):
  def __init__(self, in_controller):
    super(MainWindow, self).__init__()
    self.ui = uic.loadUi("view/main_view/main_window.ui", self)
    self.controller = in_controller
    self.ui.get_result_button.setEnabled(False)
    self.ui.clear_model_button.setEnabled(False)
    self.ui.save_model_button.setEnabled(False)

  @pyqtSlot(name='on_choose_model_button_clicked')
  def choose_model(self):
    filename, _ = QFileDialog.getOpenFileName(self, Выберите' модель для
     загрузки',
    './Data/models/', 'Model File(*.pkl)')
    if len(filename) > 0:
    status = self.controller.choose_model(filename)
    if status == Модель' успешно загружена':
      self.ui.clear_model_button.setEnabled(True)
      self.try_enable_get_result_btn()
    else:
      msgBox = QMessageBox()
      msgBox.setStandardButtons(QMessageBox.Ok)
      msgBox.setIcon(QMessageBox.Critical)
      msgBox.setText(status)
      msgBox.exec()

  @pyqtSlot(name='on_teach_model_button_clicked')
  def teach_model(self):
    self.controller.launch_teach_model_view()
    self.ui.clear_model_button.setEnabled(True)
    self.ui.save_model_button.setEnabled(True)

```

Листинг 2 – MainView Продолжение

```
@pyqtSlot(name='on_save_model_button_clicked')
def save_model(self):
  filename, _ = QFileDialog.getSaveFileName(self, Сохранить' модель',
  './Data/models/', 'Model File (*.pkl)')
  if len(filename) > 0:
    status = self.controller.save_model(filename)
    msgBox = QMessageBox()
    msgBox.setStandardButtons(QMessageBox.Ok)
    if status == Модель' успешно сохранена':
      msgBox.setIcon(QMessageBox.Information)
      msgBox.setText(status)
      self.ui.save_model_button.setEnabled(False)
    else:
      msgBox.setIcon(QMessageBox.Critical)
      msgBox.setText(status)
    msgBox.exec()

@pyqtSlot(name='on_clear_model_button_clicked')
def clear_model(self):
  self.controller.clear_model()
  self.ui.clear_model_button.setEnabled(False)
  self.ui.get_result_button.setEnabled(False)


@pyqtSlot(name='on_choose_test_audio_button_clicked')
def choose_test_audio(self):
  filename, _ = QFileDialog.getOpenFileName(self, Выберите' mp3файл- для
 тестирования',
  './Data/test/', 'Audio File(*.mp3)')
  if len(filename) > 0:
    status = self.controller.load_test_signal(filename)
    self.ui.choose_test_audio_label.setText(status)
    self.try_enable_get_result_btn()

@pyqtSlot(name='on_get_result_button_clicked')
  def get_result(self):
  self.controller.launch_result_view()

```

MainController - точка входа в приложение. Отвечает за распределение задач от *MainView* к *SoundsExtractionModel* и по агрегируемым в себе контроллерам - *ResultContoller*, *TeachModelController*.

Листинг 3 – MainController

```
class MainController:
  def __init__(self):
    self.main_window = MainWindow(self)
    self.main_window.show()
    self.sounds_extraction_module = SoundsExtractionModule()

    # инициализация teach_model_controller
    self.teach_model_controller = TeachModelController(self,
     self.sounds_extraction_module)

  # инициализация result_controller
    self.result_controller = ResultController(self,
     self.sounds_extraction_module)

  def launch_teach_model_view(self):
    self.teach_model_controller.show_window()

  def teach_model_window_closed(self):
    self.main_window.try_enable_get_result_btn()

  def launch_result_view(self):
    self.result_controller.show_window()

  def clear_model(self):
    self.sounds_extraction_module.clear_classification_model()

  def load_test_signal(self, filename):
    return self.sounds_extraction_module.load_test_signal(filename)

  def choose_model(self, path):
    return self.sounds_extraction_module.choose_model(path)

  def save_model(self, path):
    return self.sounds_extraction_module.save_classification_model(path)

```

FeatureExtractionModule - модуль, отвечающий за извлечение признаков из сигналов.

Листинг 4 – FeatureExtractionModule

```
from scipy import signal
from pywt import wavedec

from model.params.audio_params import AudioParams

class FeatureExtractionModule:

  @staticmethod
  def get_t_between_frames(frames_coeffs, offset):
    t_values = []
    cur_time = offset
    dt = AudioParams.frame_sz() * AudioParams.hop_part() / AudioParams.sr()
    for i in range(len(frames_coeffs)):
      t_values.append(cur_time)
      cur_time += dt
    return np.array(t_values)

  # получает массив сигналов: [[path, signal],...], возвращает словарь
    признаков: {path: feature}
  def extract_from_signals(self, sigs):
    res = []
    for path_sig in sigs:
      res.append([path_sig[0], self.extract_from_signal(path_sig[1])])
    return res

  # получает на вход signal, возвращает признаки signal
  def extract_from_signal(self, sig):
    # разбитие сигнала на фреймы
    frames_sig = self._splitting_signal_on_frames(sig,
    AudioParams.frame_sz(), AudioParams.hop_part())

    # устранение деффекта на краях с помощью оконной функции Хаммингa
    frames_hamming_sig = self._hamming_func(frames_sig,
    AudioParams.frame_sz())

    # применение вейвлетпреобразования-
    frames_coeffs = self._frames_wavelet(frames_hamming_sig,
    AudioParams.wavelet(), AudioParams.wavelet_level())

    return frames_coeffs

```

Листинг 5 – FeatureExtractionModule Продолжение

```
  def _splitting_signal_on_frames(self, descrete_sig, frame_sz, hop_part):
    frames = librosa.util.frame(descrete_sig, frame_length=frame_sz,
    hop_length=int(hop_part * frame_sz), axis=0)
    return frames

  def _hamming_func(self, frames_signal, frame_sz):
    w = signal.windows.hamming(frame_sz, sym=True)
    return np.array([w * frame for frame in frames_signal])

  def _wavelet_conv(self, sig, method, level):
    return np.array(wavedec(sig, method, level=level))

  def _frames_wavelet(self, frames_signal, method, level):
    return np.array([self._wavelet_conv(sig, method, level) for sig in
    frames_signal], dtype=object)


```

DataPreprocessingModule - модуль, формирующий обучающую и тестовую выборки для *ClassificationModule*.

Листинг 6 – DataPreprocessingModule

```
import numpy as np
from itertools import chain
import pandas as pd
import ast

from model.feature_extraction_module.feature_extraction_module import
    FeatureExtractionModule
from model.params.markups_params import MarkupsParams
from model.params.audio_params import AudioParams

class DataPreprocessingModule:

  # получает на вход signal, возвращает признаки signal
  def create_test_data(self, features):
    features_names =
    FeatureExtractionModule.get_wavelet_columns_names(features[0])
    res_data = []
    for feature in features:
      res_data.append(list(chain(*feature)))
    return pd.DataFrame(res_data, columns=features_names)

```

Листинг 7 – DataPreprocessingModule Продолжение

```
# получает массив признаков [[path, feature],...], разметку в pandas с колонками: ['recordname', 'start', 'end']
# возвращает обучающую выборку в pandas формате: [feature, feature, ..., y], y = {0,1}
def create_train_data(self, features, df_markups):
    # получаем названия признаков
    features_names = FeatureExtractionModule.get_wavelet_columns_names(features[0][1][0])
    features_names.append('y')
    res_data = []
    for path, frames_coeffs in features:
      filename = self._get_filename_from_path(path)
      for interval in ast.literal_eval(df_markups[df_markups[MarkupsParams.filename()] == filename]
        .intervals.iloc[0]):
        # получаем временные промежутки между фреймами
        t_positions_between_frames = FeatureExtractionModule.get_t_between_frames(frames_coeffs, interval[0])
        # формируем список Вейвлеткоэффициентов- (n) - один фрейм
        # длина( списка - t_positions_between_frames) и добавляем к нему y={0,1}
        y = np.zeros(frames_coeffs.shape[0])
        borders = self._borders_preprocessing(
        df_markups[df_markups[MarkupsParams.filename()] == filename],
        MarkupsParams.start_end()[0], MarkupsParams.start_end()[1])
        # оставляем только те границы, что лежат в интервале interval
        borders = self._filter_borders(borders, interval)
        for border in borders:
          for t_pos in range(len(t_positions_between_frames)):
          if t_positions_between_frames[t_pos] > border:
            y[t_pos] = 1.
            break
        # преобразуем frames_coeffs
        res_data_i = []
        for frame, y_i in zip(frames_coeffs, y):
          lst_i = list(chain(*frame))
          lst_i.append(y_i)
          res_data_i.append(lst_i)
        res_data.extend(res_data_i)
    return pd.DataFrame(res_data, columns=features_names)

```

ClassificationModule - модуль, отвечающий за обучение модели и осуществление ею предсказаний.

Листинг 8 – ClassificationModule

```
from model.data import Data
from model.params.classification_params import ClassificationParams

class ClassificationModule:
  def __init__(self, in_data_preprocessing_module,
   in_feature_extraction_module):
    self.data_preprocessing_module = in_data_preprocessing_module
    self.feature_extraction_module = in_feature_extraction_module
    self.params = ClassificationParams()
    self.model = DataМодель(' не выбрана')

  def clear_model(self):
    self.model = DataМодель(' не выбрана')

  def fit(self, signals, markups):
    if self.model.status == Модель' обучена':
      return self.model.status
    try:
      # получает массив сигналов: [[path, audio],...], возвращает массив
   признаков: [[path, feature],...]
      features = self.feature_extraction_module.extract_from_signals(signals)
      # получает массив признаков [[path, feature],...],
      # разметку в pandas с колонками: ['filename', 'start', 'end']
      # возвращает обучающую выборку в pandas формате: [feature, feature,
   ..., y], y = {0,1}
      train_data =
   self.data_preprocessing_module.create_train_data(features, markups)

      self.params.model.fit(train_data.drop(axis=0, columns=['y']),
   train_data['y'].astype('int64'))

      self.model.data = self.params.model
      self.model.status = Модель' обучена'
    except:
      return Ошибка' обучения модели'
      return self.model.status

```

Листинг 9 – ClassificationModule Продолжение

```
# возвращает статус и список predicted_markups временных промежутков
def predict(self, signal):
  if self.model.status != Модель' обучена':
    return self.model.status
  try:
    features = self.feature_extraction_module.extract_from_signal(signal)
    # получает на вход признаки signal, возвращает pandas признаки с их
  именами
    test_data = self.data_preprocessing_module.create_test_data(features)
    predictions = self.model.data.predict(test_data)
    return Звуки' успешно выделены',
  self._convert_predictions_to_time_list(predictions)
  except:
    return Ошибка'! Не удалось выделить звуки', None

```

3.6 Выводы

В данном разделе были выполнены следующие задачи:

- была реализована и протестировано программное обеспечение, спроектированное в разделе 2, для проверки работоспособности разработанного метода выделения звуков естественного языка в звучащей речи;
- было дано обоснование выбора языка программирования, среды разработки и используемых библиотек;
- было дано описание пользовательского интерфейса;
- были приведены листинги кода реализаций основных компонентов программного обеспечения.

4 Исследовательская часть

4.1 Формирование обучающей и тестовой выборок данных

В качестве данных для обучения и тестирования разработанного метода был задействован, размеченный студентами с кафедры Л4 и университета МГЛУ, речевой корпус. Данные представляли из себя аудио-файлы в mp3 формате и разметку к ним в формате json.

В результате выполнения предобработки данных было получено 5806 объектов по 7 признаков в каждом. Предобработка данных выполнялась при следующих начальных параметрах (раздел 2.3):

- n_{frame} = 2500;
- sr = 16000 Гц;
- hop = 40%;
- $base$ = *db4* (Вейвлет Добеши 8);
- k = 6.

Ниже приведен вывод первых пяти сформированных объектов выборки.

	A	D6	D5	D4	D3	D2	D1	y
0	0.486720	0.036786	0.010464	0.006818	0.004979	0.001184	0.000364	0
1	0.223234	0.024111	0.077241	0.060741	0.253940	0.503590	0.466350	0
2	0.704831	0.394256	0.155101	0.106348	0.133811	0.140230	0.078296	0
3	0.177642	0.124712	0.056273	0.017455	0.006439	0.003420	0.001974	0
4	0.213507	0.071184	0.017467	0.006277	0.003426	0.000703	0.000307	0

Рисунок 20 – Формат сформированной выборки.

Под тестовую выборку было решено выделить всего 20% объектов данных. Такой малый объем тестовых данных объясняется тем, что на этапе поиска оптимальных гиперпараметров модели планируется использовать кросс-валидацию [26] из 5 блоков, чтобы нейтрализовать эффект неравномерного распределения классов в тестовой выборке.

4.2 Метрики оценки качества моделей классификации

В сформированном датасете наблюдался дизбаланс классов - отношение количества объектов класса 1 (данный участок содержит в себе границу межфонемного перехода) к классу 0 равнялось$1:3$. В виду чего применение такой

метрики оценки качества модели как *доля правильных ответов алгоритма* - accuracy, будет давать неинформативные результаты.

В виду чего было решено использователь другие метрики, независящие в той или иной степени от баланса классов в выборке.

4.2.1 precision и recall

Базовыми метриками для оценки качества модели классификации с несбалансированными классами являются *precision* и *recall*:

$$precision = \frac{TP}{TP + FP} \tag{23}$$

$$recall = \frac{TP}{TP + FN} \tag{24}$$

Значения TP, FP, FN получаются из таблицы сопряженности (таблица 4.2.1), строящейся на основе результатов классификации моделью и фактической принадлежности объектов к классам.

	$y = 1$	$y = 0$
$\hat{y} = 1$	True Positive (TP)	False Positive (FP)
$\hat{y} = 0$	False Negative (FN)	True Negative (TN)

Таблица 3 – Таблица сопряженности.

- TP (True Positive) - верно классифицированные положительные объекты;
- TN (True Negative) - верно классифицированные отрицательные объекты;
- FN (False Negative) - неверно классифицированные отрицательные объекты;
- FP (False Positive) - неверно классифицированные положительные объекты.

В данной работе за положительный объект принимается класс 1 (объект содержит в себе границу межфонемного перехода), а за отрицательный - класс 0.

Метрику *precision* можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а *recall* показывает, какую долю объектов положитель-

ного класса из всех объектов положительного класса нашел алгоритм.

4.2.2 Коэффициент корреляции Мэтьюса

При оптимизации гиперпараметров метода как правило используют одну метрику, улучшение результатов которой планируется увидеть на тестовой выборке.

Одной из наиболее популярных метрик является F-мера - среднее гармоническое *precision* и *recall*:

$$F_1 = \frac{precision * recall}{precision + recall} \tag{25}$$

Однако, данная метрика все еще может дать смещенную оценку предсказания, поскольку она не учитывает показатель TN.

Наиболее оптимальным вариантом является использование *Коэффициента Корреляции Мэтьюса (ККМ)*, учитывающего все базовые показатели из таблицы сопряженности:

$$KKM = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}} \tag{26}$$

где $KKM \in [-1; +1]$.

Коэффициент +1 представляет собой идеальное предсказание, 0 не лучше, чем случайное предсказание, а -1 указывает на полное несоответствие между предсказанием и наблюдением.

4.2.3 AUC-ROC

При конвертации вероятности принадлежности объекта к классу в бинарную метку алгоритм классификации выбирает порог при котором класс 0 становится 1.

По умолчанию он равен 0.5, но такое решение далеко не всегда является оптимальным при отсутсвии баланса классов в данных.

Наиболее популярным подходом оценить модель, не привязываясь к конкретному порогу, является *AUC-ROC*, где

- *ROC* (Receiver Operating Characteristic curve) - кривая ошибок, равная отношению доли истинно положительных объектов *TPR* (True Positive Rate) к доли ложных положительных объектов *FPR* (False Positive Rate);

• *AUC* (Area Under Curve) - площадь под графиком кривой ошибок *ROC*.

При этом *TPR* и *FPR* задаются следующими формулами:

$$TPR = \frac{TP}{TP + FN}, FPR = \frac{FP}{FP + TN} \tag{27}$$

Критерий *AUC-ROC* устойчив к несбалансированным классам, и интерпретируется как вероятность того, что случайно выбранный positive объект будет будет иметь более высокую вероятность идентифицироваться классификатором как positive, чем случайно выбранный negative объект.

4.3 Поиск оптимальных гиперпараметров метода

В качестве варьируемых гиперпараметров разработанного метода были выбраны следующие характеристики (разделы 2.3 и 2.5):

- *k* - количество отсчетов (замеров величины сигнала) в одном фрейме;
- C - параметр регуляризации модели классификации;
- *class_weight* - параметр, позволяющий варьировать размер штрафа при неудачном выборе класса в модели классификации.

При этом $kernel$ - ядро SVM, задающее функцию разделяющей гиперплоскости, примает значение *rbf*, $gamma$ - ширина ядра, принимает значение *scale* по умолчанию.

4.3.1 GridSearchCV с применением ККМ

Подбор гиперпараметров C и *class_weight* осуществлялся путем применения инструмента *GridSearchCV* из библиотеки sklearn, реализующий внутри себя кросс-валидацию. В качестве метрики качества был использован *Коэффициент Корреляции Мэтьюса (раздел 4.2.2)*.

Интервал варьирования гиперпараметра C равнялся $[10, 10^6]$, а гиперпараметр *class_weight* принимал значение $None$ либо $balanced$.

Ниже приведена таблица усредненных на всех блоках результатов применения *GridSearchCV* для гиперпараметров C и *class_weight*:

	class_weight = *None*	class_weight = *balanced*
$C = 10$	0.290	0.274
$C = 10^2$	0.302	0.293
$C = 10^3$	0.313	0.307
$C = 7 * 10^3$	0.318	0.318
$C = 8 * 10^3$	0.316	0.314
$C = 9 * 10^3$	0.318	0.317
$C = 10^4$	0.317	0.313
$C = 2 * 10^4$	0.320	0.307
$C = 3 * 10^4$	0.318	0.306
$C = 4 * 10^4$	0.318	0.303
$C = 10^5$	0.324	0.304
$C = 10^6$	0.307	0.293

Таблица 4 – Результаты *GridSearchCV*. Метрика качества - *Коэффициент Корреляции Мэтьюса*.

Из приведенной выше таблицы следует, что наиболее оптимальными значениями гиперпараметров C и *class_weight* являются 10^5 и *None*, соответственно.

Подбор гиперпараметра *k* осуществлялся вне GridSearchCV из-за необходимости проведения слишком большого количества экспериментов.

Для исследования влияния параметра *k* на разработанный метод, было проведено 7 экспериментов обучения и тестирования метода при $k = 4, 6, 8, 10, 12, 14, 16$. Для проведения исследования были выбраны наиболее оптимальные значения гиперпараметров C и *class_weight*, полученных в предыдущем шаге.

Ниже приведен график зависимости *Коэффициента Корреляции мэтьюса ККМ* от параметра *k*.

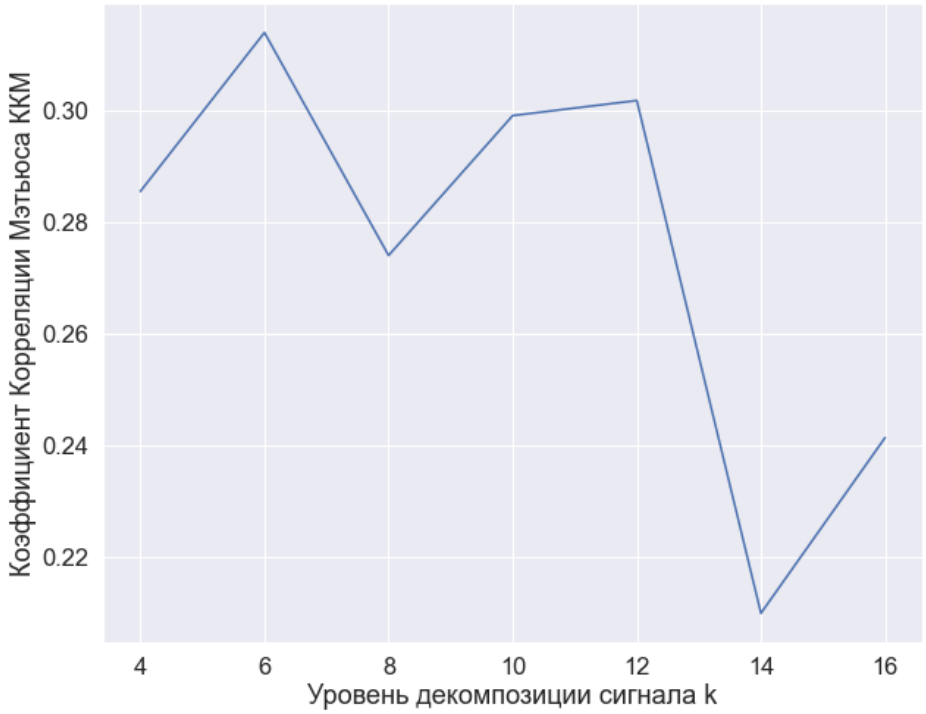

Рисунок 21 – График зависимости*Коэффициента Корреляции Мэтьюса ККМ* от уровня декомпозиции сигнала k.

Из приведенного выше графика видно, что наибольшее значение параметр *ММК* принимает при $k = 6$.

4.4 Оценка качества модели классификации

Ниже приведены результаты применения описанных выше метрик на обученной модели классификации при выборе оптимальных параметров:

	precision	recall
positive	0.72	0.56
negative	0.63	0.77

Таблица 5 – Точность (precision) и полнота (recall) предсказания модели.

Коэффициент Корреляции Мэтьюса (ККМ) равен 0.33.

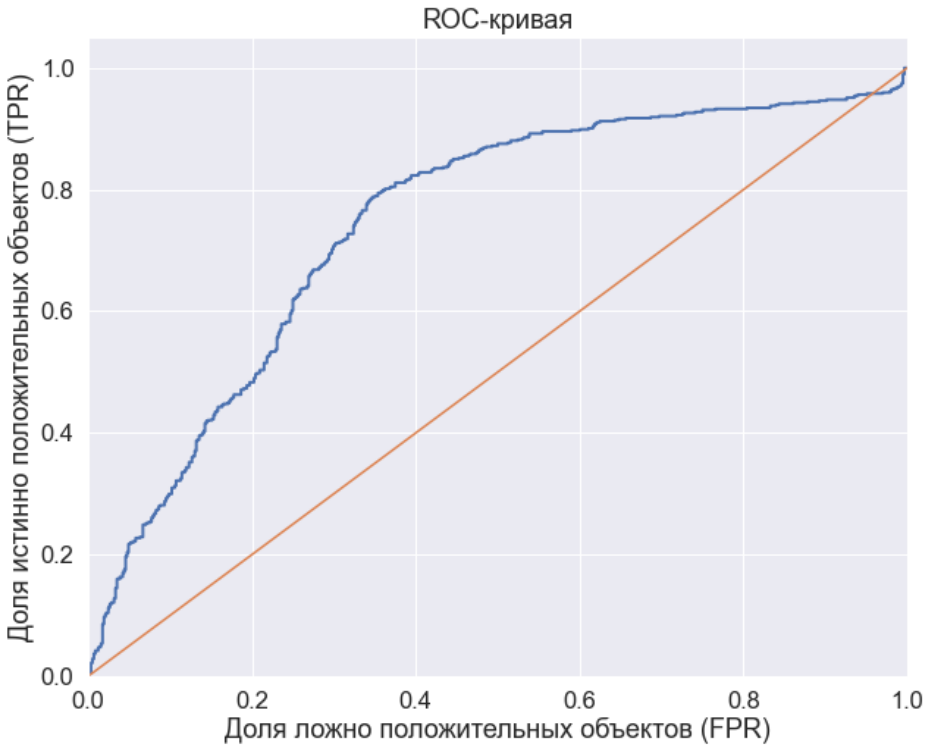

Рисунок 22 – ROC-кривая.

При этом $AUC = 0.7325$.

В литературе [27] приводится следующая экспертная шкала для значений AUC, по которой можно судить о качестве модели:

Интервал AUC	Качество модели
0.9-1.0	отличное
0.8-0.9	очень хорошее
0.7-0.8	хорошее
0.6-0.7	среднее
0.5-0.6	удовлетворительное

Таблица 6 – Экспертная шкала значений AUC.

Откуда следует, что полученная модель имеет хорошее качество предска-

зания.

4.5 Выводы

В результате проведения исследовательской работы над разработанным методом были выполнены следующие задачи:

- были сформированы обучающие и тестовые выборки данных для обучения модели классификации обнаружения межфонемных переходов;
- были рассмотрены основные метрики оценки качества модели классификации применительно к данной задаче, а именно, к задаче с несбалансированными классами;
- был произведен поиск оптимальных гиперпараметров метода при помощи Коэффициента корреляции Мэтьюса (ККМ);
- была произведена оценка качества модели на оптимальных гиперпараметрах при помощи рассмотренных метрик.

Таким образом, полученная модель имеет хорошее качество предсказания, что означает, что разработанный метод может быть применим для решения задачи сегментации речевого сигнала.

Такое относительно невысокое качество распознавания обусловлено малым размером обучающей выборки и сильной погрешностью измерений при разметке сформированных данных. Это связано с субъективностью человеческого слухового восприятия - все звуки размечались вручную разными людьми в результате чего возникла большая погрешность измерений.

ЗАКЛЮЧЕНИЕ

В результате выполнения выпускной квалификационной работы были успешно достигнуты все поставленные задачи:

- проанализирована предметная область:
  - описаны основные этапы распознавания речевого сигнала;
  - обоснована актуальность задачи сегментации речевого сигнала;
  - проведен анализ существующих решений сегментации речевого сигнала.
- проведен анализ существующих методов выделения признаков речевого сигнала и алгоритмов классификации;
- выбраны методы наиболее оптимальные для поставленной цели;
- на их основе спроектирован метод выделения звуков естественного языка в звучащей речи и разработан алгоритм, реализующий данный метод;
- спроектирована систему для проверки работоспособности метода;
- сформирована выборка для проведения эксперимента;
- проведено исследование разработанного метода на данной выборке;

Проведя исследование разработанного метода, выяснилось, что разработанный метод может быть применим для решения задачи сегментации речевого сигнала. Однако, реализованная модель имеет относительно невысокое качество предсказания, что обусловлено малым размером обучающей выборки и сильной погрешностью измерений при разметке сформированных данных.

Это связано с субъективностью человеческого слухового восприятия - все звуки размечались вручную разными людьми в результате чего возникла большая погрешность измерений.

Можно обозначить следующие пути развития данной работы:

- подготовка большего количества обучающей выборки, имеющей меньшую погрешность измерений для повышения точности метода;
- применение альтернативных алгоритмов классификации, позволяющими повысить качество распознавания с применением большего количества данных.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Аграновский А.В., Леднов Д.А., Телеснин Б.А. Сегментация речи (математическая модель) // Информационные технологии. 1998. № 9. С. 24-28

2. Казачкин, А. Е. Методы распознавания речи, современные речевые технологии [Электронный ресурс]. - Режим доступа: https://moluch.ru/archive/277/62675/, свободный - (17.03.2021)

3. Шпаков, Д. В. Распознавание голоса в сфере информационных технологий [Электронный ресурс]. - Режим доступа: https://moluch.ru/archive/163/45163/, свободный - (17.03.2021)

4. Федосин С.А., Еремин А. Ю. Классификация систем распознавания речи. [Электронный ресурс]. - Режим доступа: http://fetmag.mrsu.ru/2010-2/pdf/SpeechRecognition.pdf, свободный - (19.04.2021)

5. Тампель И.Б, Карпов А.А. Автоматическое распознавание речи. — СПб. : Университет ИТМО, 2016. — С. 113. [Электронный ресурс]. - Режим доступа: https://books.ifmo.ru/file/pdf/1921.pdf, свободный - (19.04.2021)

6. Рамишвили Г. С. Автоматическое опознавание говорящего по голосу. М. : Радио и связь, 1981. 224 с

7. О.А. Вишнякова, Д.Н. Лавров АВТОМАТИЧЕСКАЯ СЕГМЕНТАЦИЯ РЕЧЕВОГО СИГНАЛА НА БАЗЕ ДИСКРЕТНОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ [Электронный ресурс]. - Режим доступа: http://msm.omsu.ru/jrns/jrn23/VishnyakovaLavrov.pdf, свободный - (22.04.2021)

8. Т. Ермоленко, В. Шевчук АЛГОРИТМЫ СЕГМЕНТАЦИИ С ПРИМЕНЕНИЕМ БЫСТРОГО ВЕЙВЛЕТ-ПРЕОБРАЗОВАНИЯ [Электронный ресурс]. - Режим доступа: http://www.dialog-21.ru/media/2715/ermolenko.pdf, свободный - (22.04.2021)

9. О.А. Вишнякова, Д.Н. Лавров ПРИМЕНЕНИЕ ПРЕОБРАЗОВАНИЯ ГИЛЬБЕРТА-ХУАНГА К ЗАДАЧЕ СЕГМЕНТАЦИИ РЕЧИ [Электронный ресурс]. - Режим доступа:

http://msm.omsu.ru/jrns/jrn24/Vishnakova_n24_2011.pdf, свободный - (10.05.2021)

10. И.В. Дашкевич. МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ Модуль голосовой идентификации диктора [Электронный ресурс]. - Режим доступа: http://elib.sfukras.ru/bitstream/handle/2311/74791/diplom_magistratura.pdf?sequence=1, свободный - (23.04.2021)

11. Музычук, Д. С. Использование преобразования Гильберта-Хуанга для формирования моделей фонем русского языка в задаче распознавания речи [Электронный ресурс]. - Режим доступа: https://moluch.ru/archive/53/7041/, свободный - (15.05.2021).

12. Давыдов А. В. Цифровая обработка сигналов: Тематические лекции. — Екатеринбург: УГГУ, ИГиГ, ГИН, Фонд электронных документов, 2005.

13. Томчук Кирилл Константинович, Сегментация речевых сигналов для задач автоматической обработки речи [Электронный ресурс]. - Режим доступа: https://fs.guap.ru/dissov/tomchuk_kk/full.pdf, свободный - (10.05.2021)

14. Т.В. Батура, МЕТОДЫ АВТОМАТИЧЕСКОЙ КЛАССИФИКАЦИИ ТЕКСТОВ [Электронный ресурс]. - Режим доступа: https://www.researchgate.net/publication/315328102_Metody_avtomaticeskoj_klassifikacii_tekstov, свободный - (11.05.2021)

15. К.В. Воронцов Лекции по методу опорных векторов [Электронный ресурс]. - Режим доступа: http://www.ccas.ru/voron/download/SVM.pdf, свободный - (11.05.2021)

16. Осциллографирование электрических сигналов цифровым осциллографом [Электронный ресурс]. - Режим доступа: https://portal.tpu.ru/SHARED/k/KOZHEMYAK/Teaching/Tab1/Electronics/Lab_1_Digital_oscilloscope_2018.pdf, свободный - (10.05.2021)

17. Python documentation [Электронный ресурс]. - Режим доступа: https://www.python.org/doc/, свободный - (11.05.2021)

18. Pycharm documentation [Электронный ресурс]. - Режим доступа: https://www.jetbrains.com/help/pycharm/inline-documentation.html, свободный - (11.05.2021)

19. Librosa documentation [Электронный ресурс]. - Режим доступа: https://librosa.org/doc/latest/index.html, свободный - (11.05.2021)

20. PyWavelets documentation [Электронный ресурс]. - Режим доступа: https://pywavelets.readthedocs.io/_/downloads/en/v1.0.0/pdf/, свободный - (11.05.2021)

21. Scikit-learn documentation [Электронный ресурс]. - Режим доступа: https://www.sklearn.org/documentation.html, свободный - (11.05.2021)

22. Pandas documentation [Электронный ресурс]. - Режим доступа: https://pandas.pydata.org/pandas-docs/stable/index.html, свободный - (11.05.2021)

23. Matplotlib documentation [Электронный ресурс]. - Режим доступа: https://matplotlib.org/, свободный - (11.05.2021)

24. NumPy documentation [Электронный ресурс]. - Режим доступа: https://numpy.org/doc/, свободный - (11.05.2021)

25. PyQt5 documentation [Электронный ресурс]. - Режим доступа: https://doc.qt.io/qtforpython/, свободный - (11.05.2021)

26. Кросс-валидация [Электронный ресурс]. - Режим доступа: http://datascientist.one/cross-validation/, свободный - (06.04.2021)

27. Zweig, Mark H.; Campbell, Gregory. Receiver-operating characteristic (ROC) plots: a fundamental evaluation tool in clinical medicine. Clinical Chemistry. Нью-Йорк: Изд-во Oxford, 577 с.

ПРИЛОЖЕНИЕ А

Ниже приведены листинги разработанного ПО.

MainView

```
from PyQt5 import uic
from PyQt5.QtCore import pyqtSlot, Qt
from PyQt5.QtWidgets import QMessageBox, QMainWindow, QFileDialog


class MainWindow(QMainWindow):
    def __init__(self, in_controller):
        super(MainWindow, self).__init__()
        self.ui = uic.loadUi("view/main_view/main_window.ui", self)
        self.controller = in_controller
        self.ui.get_result_button.setEnabled(False)
        self.ui.clear_model_button.setEnabled(False)
        self.ui.save_model_button.setEnabled(False)

    @pyqtSlot(name='on_choose_model_button_clicked')
    def choose_model(self):
        filename, _ = QFileDialog.getOpenFileName(self, Выберите' модель для загрузки',
                                                  './Data/models/', 'Model File(*.pkl)')
        if len(filename) > 0:
            status = self.controller.choose_model(filename)
            if status == Модель' успешно загружена':
                self.ui.teach_model_button.setEnabled(False)
                self.ui.choose_model_button.setEnabled(False)
                self.ui.clear_model_button.setEnabled(True)
                self.ui.choose_model_label.setTextМодель(' выбрана')
                self.try_enable_get_result_btn()
            else:
                msgBox = QMessageBox()
                msgBox.setStandardButtons(QMessageBox.Ok)
                msgBox.setIcon(QMessageBox.Critical)
                msgBox.setText(status)
                msgBox.exec()

    @pyqtSlot(name='on_teach_model_button_clicked')
    def teach_model(self):
        self.controller.launch_teach_model_view()
        self.ui.clear_model_button.setEnabled(True)
        self.ui.save_model_button.setEnabled(True)
```

MainView

```
1 ВыберитемодельдлязагрузкиМодельуспешнозагруженаМодельвыбрана
2
3     @pyqtSlot(name='on_save_model_button_clicked')
4     def save_model(self):
5         filename, _ = QFileDialog.getSaveFileName(self, Сохранить' модель',
6                                                   './Data/models/', 'Model
      File (*.pkl)')
7         if len(filename) > 0:
8             status = self.controller.save_model(filename)
9             msgBox = QMessageBox()
10            msgBox.setStandardButtons(QMessageBox.Ok)
11            if status == Модель' успешно сохранена':
12                msgBox.setIcon(QMessageBox.Information)
13                msgBox.setText(status)
14                self.ui.save_model_button.setEnabled(False)
15            else:
16                msgBox.setIcon(QMessageBox.Critical)
17                msgBox.setText(status)
18            msgBox.exec()
19
20    @pyqtSlot(name='on_clear_model_button_clicked')
21    def clear_model(self):
22        self.controller.clear_model()
23        self.ui.clear_model_button.setEnabled(False)
24        self.ui.save_model_button.setEnabled(False)
25        self.ui.get_result_button.setEnabled(False)
26        self.ui.teach_model_button.setEnabled(True)
27        self.ui.choose_model_button.setEnabled(True)
28
29
30    @pyqtSlot(name='on_choose_test_audio_button_clicked')
31    def choose_test_audio(self):
32        filename, _ = QFileDialog.getOpenFileName(self, Выберите' mp3файл-
      для тестирования',
33                                                  './Data/test/', 'Audio
      File(*.mp3)')
34        if len(filename) > 0:
35            status = self.controller.load_test_signal(filename)
36            self.ui.choose_test_audio_label.setText(status)
37            self.try_enable_get_result_btn()
38
39    @pyqtSlot(name='on_get_result_button_clicked')
40    def get_result(self):
41        self.controller.launch_result_view()
```

ResultView

```
from PyQt5 import uic
from PyQt5.QtCore import pyqtSlot, Qt
from PyQt5.QtGui import QPixmap
from PyQt5.QtWidgets import QMessageBox, QWidget


class ResultWindow(QWidget):
    def __init__(self, in_controller):
        super(ResultWindow, self).__init__()
        self.ui = uic.loadUi("view/result_view/result_window.ui", self)
        self.controller = in_controller

    @pyqtSlot(name='on_close_button_clicked')
    def close_btn(self):
        self.close()

    def paint_graph(self):
        status, result_graph_path = self.controller.extract_sounds()
        if len(result_graph_path) > 0:
            self.ui.graph_label.setStyleSheet("QLabel{{border-image:
    url({});}}"
                                              .format(result_graph_path))
        else:
            msgBox = QMessageBox()
            msgBox.setStandardButtons(QMessageBox.Ok)
            msgBox.setIcon(QMessageBox.Critical)
            msgBox.setText(status)
            msgBox.exec()
```

TeachModelView

```
from PyQt5 import uic
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtWidgets import QMessageBox, QDialog, QFileDialog


class TeachModelWindow(QDialog):
    def __init__(self, in_controller):
        super(TeachModelWindow, self).__init__()
        self.ui = uic.loadUi("view/teach_model_view/teach_model_window.ui", self)
        self.controller = in_controller
        self.ui.fit_model_button.setEnabled(False)
        self.ui.choose_fit_audio_button.setEnabled(False)

    @pyqtSlot(name='on_close_button_clicked')
    def close_btn(self):
        self.controller.window_closed()
        self.close()

    @pyqtSlot(name='on_choose_fit_audio_button_clicked')
    def choose_fit_audio(self):
        filenames, _ = QFileDialog.getOpenFileNames(self, 'Choose mp3-files for learning',
                                                    './Data/train/', 'Audio Files(*.mp3)')
        if len(filenames) > 0:
            status = self.controller.load_train_signals(filenames)
            self.ui.choose_fit_audio_label.setText(status)

            markups_status = self.controller.get_status('train_markups')
            if status == 'Audio-files have chosen' and markups_status == 'Markups have chosen':
                model_status = self.controller.get_status('model')
                if model_status == 'Model havent chosen':
                    self.ui.fit_model_button.setEnabled(True)

    @pyqtSlot(name='on_fit_model_button_clicked')
    def fit_model_btn(self):
        status = self.controller.fit_model()
        msgBox = QMessageBox()
        msgBox.setStandardButtons(QMessageBox.Ok)
        if status == 'Model have chosen':
            msgBox.setIcon(QMessageBox.Information)
            msgBox.setText(status)
```

TeachModelView

```
            self.ui.fit_model_label.setText(status)
        else:
            msgBox.setIcon(QMessageBox.Critical)
            msgBox.setText(status)
        msgBox.exec()

    @pyqtSlot(name='on_choose_fit_markups_button_clicked')
    def choose_fit_markups(self):
        filename, _ = QFileDialog.getOpenFileName(self, 'Choose csv-markups for learning',
                                                  './Data/train/', 'CSV File(*.csv)')
        if len(filename) > 0:
            status = self.controller.load_train_markups(filename)
            self.ui.choose_fit_markups_label.setText(status)

            if status == 'Markups have chosen':
                self.ui.choose_fit_audio_button.setEnabled(True)
                audios_status = self.controller.get_status('train_signals')
                if audios_status == 'Audio-files have chosen':
                    model_status = self.controller.get_status('model')
                    if model_status == 'Model havent chosen':
                        self.ui.fit_model_button.setEnabled(True)
```

MainController

```

from view.main_view.main_window import MainWindow
from view.result_view.result_window import ResultWindow
from view.teach_model_view.teach_model_window import TeachModelWindow

from ..result_controller.result_controller import ResultController
from ..teach_model_controller.teach_model_controller import TeachModelController

from model.sounds_extraction_module.sounds_extraction_module import SoundsExtractionModule
```

MainController

```
1
2 class MainController:
3     def __init__(self):
4         self.main_window = MainWindow(self)
5         self.main_window.show()
6         self.sounds_extraction_module = SoundsExtractionModule()
7
8         # инициализация teach_model_controller
9         self.teach_model_controller = TeachModelController(self,
      self.sounds_extraction_module)
10
11        # инициализация result_controller
12        self.result_controller = ResultController(self,
      self.sounds_extraction_module)
13
14    def launch_teach_model_view(self):
15        self.teach_model_controller.show_window()
16
17    def teach_model_window_closed(self):
18        self.main_window.try_enable_get_result_btn()
19
20    def launch_result_view(self):
21        self.result_controller.show_window()
22
23    def clear_model(self):
24        self.sounds_extraction_module.clear_classification_model()
25
26    def load_test_signal(self, filename):
27        return self.sounds_extraction_module.load_test_signal(filename)
28
29    def get_status(self, data):
30        if data == 'test_signal':
31            return self.sounds_extraction_module.get_status_test_signal()
32        elif data == 'model':
33            return
      self.sounds_extraction_module.get_status_classification_module()
34
35    def choose_model(self, path):
36        return self.sounds_extraction_module.choose_model(path)
37
38    def save_model(self, path):
39        return self.sounds_extraction_module.save_classification_model(path)
```

ResultController

```
1
2 from view.result_view.result_window import ResultWindow
3
4
5 class ResultController:
6     def __init__(self, in_main_controller, in_sounds_extraction_module):
7         self.main_controller = in_main_controller
8         self.sounds_extraction_module = in_sounds_extraction_module
9         self.window = ResultWindow(self)
10
11     def show_window(self):
12         self.window.show()
13         self.window.paint_graph()
14
15     def extract_sounds(self):
16         return self.sounds_extraction_module.predict_classification_model()
```

TeachModelController

```
1
2 from view.teach_model_view.teach_model_window import TeachModelWindow
3
4 class TeachModelController:
5     def __init__(self, in_main_controller, in_sounds_extraction_module):
6         self.main_controller = in_main_controller
7         self.window = TeachModelWindow(self)
8         self.sounds_extraction_module = in_sounds_extraction_module
9
10     def show_window(self):
11         self.window.show()
12
13     def window_closed(self):
14         self.main_controller.teach_model_window_closed()
15
16     def load_train_signals(self, filenames):
17         return self.sounds_extraction_module.load_train_signals(filenames)
18
19     def load_train_markups(self, filename):
20         return self.sounds_extraction_module.load_train_markups(filename)
21
22     def get_status(self, data):
23         if data == 'train_signals':
24             return self.sounds_extraction_module.get_status_train_signals()
```

ClassificationModule

```
1
2 from model.data import Data
3 from model.params.classification_params import ClassificationParams
4
5 class ClassificationModule:
6     def __init__(self, in_data_preprocessing_module,
      in_feature_extraction_module):
7         self.data_preprocessing_module = in_data_preprocessing_module
8         self.feature_extraction_module = in_feature_extraction_module
9         self.params = ClassificationParams()
10        self.model = DataМодель(' не выбрана')
11
12    def clear_model(self):
13        self.model = DataМодель(' не выбрана')
14
15    def fit(self, signals, markups):
16        if self.model.status == Модель' обучена':
17            return self.model.status
18        try:
19            features =
      self.feature_extraction_module.extract_from_signals(signals)
20
21            train_data =
      self.data_preprocessing_module.create_train_data(features, markups)
22
23
24            train_data = train_data.sample(frac=1)
25
26            y_train = train_data['y'].astype('int64')
27            X_train = train_data.drop(axis=0, columns=['y'])
28
29            self.params.model.fit(X_train, y_train)
30
31            self.model.data = self.params.model
32            self.model.status = Модель' обучена'
33        except:
34            return Ошибка' обучения модели'
35        return self.model.status
36
37
38    def predict(self, signal):
39        if self.model.status != 'Model have teached':
```

ClassificationModule

```
МодельневыбранаМодельневыбранаМодельобученаМодельобученаОшибкаобученияМодели
            return self.model.status
        try:
            features =
    self.feature_extraction_module.extract_from_signal(signal)

            test_data =
    self.data_preprocessing_module.create_test_data(features)

            test_data = test_data.sample(frac=1)
            predictions = self.model.data.predict(test_data)
            return 'Sounds have got',
    self._convert_predictions_to_time_list(predictions)
        except:
            return 'Error! Cant get sounds.', None


     def _convert_predictions_to_time_list(self, predictions):

        time_list =
    self.feature_extraction_module.get_t_between_frames(predictions, 0.)

        res = []
        dt = time_list[1] - time_list[0]
        for t, prediction in zip(time_list, predictions):
            if prediction:
                res.append(t + dt)
        return res

     def load_model(self, model):
        self.model.data = model
        self.model.status = 'Model have chosen'
        return 'Model loaded'
```

DataPreprocessingModule

```
1 import numpy as np
2 from itertools import chain
3 import pandas as pd
4 import ast
5
6 from model.feature_extraction_module.feature_extraction_module import
      FeatureExtractionModule
7 from model.params.markups_params import MarkupsParams
8 from model.params.audio_params import AudioParams
9
10 class DataPreprocessingModule:
11
12
13     def create_test_data(self, features):
14         features_names =
      FeatureExtractionModule.get_wavelet_columns_names(features[0])
15         #res_data = []
16         #for feature in features:
17         #    res_data.append(list(chain(*feature)))
18         #return pd.DataFrame(res_data, columns=features_names)
19         return pd.DataFrame(features, columns=features_names)
20
21     def create_train_data(self, features, df_markups):
22
23         features_names =
      FeatureExtractionModule.get_wavelet_columns_names(features[0][1][0])
24         features_names.append('y')
25         res_data = []
26         for path, frames_coeffs in features:
27             filename = self._get_filename_from_path(path)
28             for interval in
      ast.literal_eval(df_markups[df_markups[MarkupsParams.filename()] ==
      filename]
29                                                     .intervals.iloc[0]):
30
31                 t_positions_between_frames =
      FeatureExtractionModule.get_t_between_frames(frames_coeffs, interval[0])
32
33                 y = np.zeros(frames_coeffs.shape[0])
34                 borders = self._borders_preprocessing(
35                     df_markups[df_markups[MarkupsParams.filename()] ==
      filename],
36                     MarkupsParams.start_end()[0],
      MarkupsParams.start_end()[1])
```

DataPreprocessingModule

```
1
2                borders = self._filter_borders(borders, interval)
3                for border in borders:
4                    for t_pos in range(len(t_positions_between_frames)):
5                        if t_positions_between_frames[t_pos] > border:
6                            y[t_pos] = 1.
7                            break
8
9                res_data_i = []
10               for frame, y_i in zip(frames_coeffs, y):
11                   #lst_i = list(chain(*frame))
12                   lst_i = frame
13                   lst_i.append(y_i)
14                   res_data_i.append(lst_i)
15               res_data.extend(res_data_i)
16       return pd.DataFrame(res_data, columns=features_names)
17
18
19   def _filter_borders(self, borders, interval):
20       res = []
21       for border in borders:
22           if border >= interval[0] and border <= interval[1]:
23               res.append(border)
24       return res
25
26   def _get_filename_from_path(self, path):
27       pos = path.rfind('/')
28       if pos != -1:
29           return path[pos + 1:]
30       return path
31
32   def _borders_preprocessing(self, df, column_name_1, column_name_2):
33
34       borders = np.sort(np.array(self._merge_2_columns_on_row(df,
     column_name_1, column_name_2)))
35
36
37       return self._check_on_min_phonem_interval(borders,
     AudioParams.min_phonem_interval())
38
39   def _merge_2_columns_on_row(self, df, column_name_1, column_name_2):
40       lst = []
```

FeatureExtractionModule

```
import numpy as np
import librosa
from scipy import signal
from pywt import wavedec

from model.params.audio_params import AudioParams

class FeatureExtractionModule:

    @staticmethod
    def get_wavelet_columns_names(features):
        '''columns = []
        n = len(features) - 1
        for pos, A_coeffs in enumerate(features[0]):
            columns.append('A_' + str(n) + str(pos))
        for i, D_coeffs in enumerate(features[1:]):
            for j, D_i_coeffs in enumerate(D_coeffs):
                columns.append('D_' + str(n - i) + '_' + str(j))
        return columns'''
        #return ['A', 'D6', 'D5', 'D4', 'D3', 'D2', 'D1']
        res = ['D' + str(i) for i in range(len(features) - 1)]
        res.insert(0, 'A')
        return res
        #return ['D' + str(i) for i in range(len(features))]

    @staticmethod
    def get_t_between_frames(frames_coeffs, offset):
        t_values = []
        cur_time = offset
        dt = AudioParams.frame_sz() * AudioParams.hop_part() /
    AudioParams.sr()
        for i in range(len(frames_coeffs)):
            t_values.append(cur_time)
            cur_time += dt
        return np.array(t_values)

    @staticmethod
    def get_t_between_pos(sig, offset):
        t_values = []
        cur_time = offset
        dt = 1 / AudioParams.sr()
        for i in range(len(sig)):
            t_values.append(cur_time)
            cur_time += dt
        return np.array(t_values)
```

FeatureExtractionModule

```
        return np.array(t_values)

    # получает массив сигналов: [[path, signal],...], возвращает словарь признаков: {path: feature}
    def extract_from_signals(self, sigs):
        res = []
        for path_sig in sigs:
            res.append([path_sig[0], self.extract_from_signal(path_sig[1])])
        return res

    # получает на вход signal, возвращает признаки signal
    def extract_from_signal(self, sig):
        # разбитие сигнала на фреймы
        frames_sig = self._splitting_signal_on_frames(sig, AudioParams.frame_sz(), AudioParams.hop_part())

        # устранение деффекта на краях с помощью оконной функции Хамминга
        frames_hamming_sig = self._hamming_func(frames_sig, AudioParams.frame_sz())

        # применение вейвлетпреобразования-
        frames_coeffs = self._frames_wavelet(frames_hamming_sig, AudioParams.wavelet(), AudioParams.wavelet_level())

        return frames_coeffs

    def _splitting_signal_on_frames(self, descrete_sig, frame_sz, hop_part):
        frames = librosa.util.frame(descrete_sig, frame_length=frame_sz, hop_length=int(hop_part * frame_sz), axis=0)
        return frames

    def _hamming_func(self, frames_signal, frame_sz):
        w = signal.windows.hamming(frame_sz, sym=True)
        return np.array([w * frame for frame in frames_signal])

    def _wavelet_conv(self, sig, method, level):
        levels = np.array(wavedec(sig, method, level=level))
        return [np.square(level).sum() for level in levels]

    def _frames_wavelet(self, frames_signal, method, level):
        return np.array([self._wavelet_conv(sig, method, level) for sig in frames_signal], dtype=object)
```

GraphModule

```
import matplotlib.pyplot as plt

class GraphModule:
    # получает список временных промежутков и дискретный сигнал. Возвращает график
    def create_predicted_graph(self, predicted_markups, test_signal, t_lst):
        fig = plt.figure(figsize=(20, 7))
        ax = plt.plot(t_lst, test_signal)
        plt.vlines(predicted_markups, test_signal.min(), test_signal.max(), color='r')
        plt.xlabelВремя(', c')
        plt.ylabelАмплитуда('')
        #plt.legend([name])
        #plt.title(name)
        return fig
```

LoadDataModule

```
import pandas as pd
import librosa
from itertools import chain
import ast
import matplotlib.pyplot as plt
import pickle

from model.data import Data
from model.params.audio_params import AudioParams
from model.params.markups_params import MarkupsParams


class LoadDataModule:
    def __init__(self):
        self.test_signal = Data('Audio-file havent chosen')
        self.test_signal_filename = 'undefined_filename'
        self.train_signals = Data('Audio-files havent chosen')
        self.train_markups = Data('Markups havent chosen')


    def load_train_signals(self, paths):
        res = []
        for path in paths:
            df = self.train_markups.data
            filename = self._get_filename_from_path(path)
```

LoadDataModule

```
            intervals = ast.literal_eval(df[df.recordname ==
filename].intervals.iloc[0])
            for interval in intervals:
                res.append([path, librosa.load(path, sr=AudioParams.sr(),
offset=interval[0],

duration=interval[1]-interval[0])[0]])
        self.train_signals.data = res
        self.train_signals.status = 'Audio-files have chosen'
        return self.train_signals.status

    def _get_filename_from_path(self, path):
        pos = path.rfind('/')
        if pos != -1:
            return path[pos + 1:]
        return path


    def load_train_markups(self, path):
        df = pd.read_csv(path)
        self.train_markups.data = df
        self.train_markups.status = 'Markups have chosen'
        return self.train_markups.status


    def load_test_signal(self, path):
        self.test_signal_filename = self._get_filename_from_path(path)[:-4]
        self.test_signal.data, _ = librosa.load(path, sr=AudioParams.sr(),
offset=0.)
        self.test_signal.status = 'Audio-file have chosen'
        return self.test_signal.status


    def save_predicted_markups(self, predicted_markups):
        df = pd.DataFrame({'y': predicted_markups})
        df.to_csv('./Data/result_markups/'+self.test_signal_filename+'.csv')


    def save_predicted_graph(self, graph):
        graph_path = './Data/result_photo/'+self.test_signal_filename+'.png'
        plt.savefig(graph_path)
        return graph_path
```

LoadDataModule

```
1
2     def clear_model_data(self):
3         self.train_signals = Data('Audio-files havent chosen')
4         self.train_markups = Data('Markups havent chosen')
5
6     def load_model(self, path):
7         with open(path, 'rb') as file:
8             return pickle.load(file)
9
10    def save_model(self, model, path):
11        with open(path, 'wb') as file:
12            pickle.dump(model, file)
13        return 'Model saved'
```

SoundsExtractionModule

```
1
2 from model.load_data_module.load_data_module import LoadDataModule
3 from model.classification_module.classification_module import
      ClassificationModule
4 from model.feature_extraction_module.feature_extraction_module import
      FeatureExtractionModule
5 from model.data_preprocessing_module.data_preprocessing_module import
      DataPreprocessingModule
6 from model.graph_module.graph_module import GraphModule
7
8
9 class SoundsExtractionModule:
10    def __init__(self):
11        self.load_data_module = LoadDataModule()
12        self.data_preprocessing_module = DataPreprocessingModule()
13        self.feature_extraction_module = FeatureExtractionModule()
14        self.graph_module = GraphModule()
15        self.classification_module =
      ClassificationModule(self.data_preprocessing_module,
16
      self.feature_extraction_module)
17
18
19    def get_status_classification_module(self):
20        return self.classification_module.model.status
21
22
23    def get_status_test_signal(self):
```

SoundsExtractionModule

```
        return self.load_data_module.test_signal.status


    def get_status_train_signals(self):
        return self.load_data_module.train_signals.status


    def get_status_train_markups(self):
        return self.load_data_module.train_markups.status

    def clear_classification_model(self):
        self.load_data_module.clear_model_data()
        self.classification_module.clear_model()


    def fit_classification_model(self):
        if self.get_status_classification_module() != 'Model havent chosen':
            return 'Model chosen'
        train_signals_status = self.get_status_train_signals()
        if train_signals_status == 'Audio-files havent chosen':
            return train_signals_status
        train_markups_status = self.get_status_train_markups()
        if train_markups_status == 'Markups havent chosen':
            return train_markups_status
        status =
    self.classification_module.fit(self.load_data_module.train_signals.data,

    self.load_data_module.train_markups.data)
        return status

    def predict_classification_model(self):
        if self.get_status_classification_module() != 'Model teached':
            return 'Model didnt chosen', ''
        test_signal_status = self.get_status_test_signal()
        if test_signal_status == 'Audio-files havent chosen':
            return test_signal_status, ''
        # возвращает статус и список predicted_markups временных промежутков
        status, predicted_markups =
    self.classification_module.predict(self.load_data_module.test_signal.data)
        if predicted_markups:
            self.load_data_module.save_predicted_markups(predicted_markups)
            graph =
    self.graph_module.create_predicted_graph(predicted_markups,
    self.load_data_module.test_signal.data,
```

SoundsExtractionModule

```
возвращаетстатусисписоквременныхпромежутков

    self.feature_extraction_module.get_t_between_pos(

    self.load_data_module.test_signal.data, 0.))
            graph_path = self.load_data_module.save_predicted_graph(graph)
            return status, graph_path
        else:
            return status, ''


    def save_classification_model(self, path):
        classification_status = self.get_status_classification_module()
        if classification_status != 'Model teached':
            return 'Model havent chosen'
        return
    self.load_data_module.save_model(self.classification_module.model.data,
    path)


    def load_train_signals(self, filenames):
        status = self.get_status_train_signals()
        if status == 'Audio-files have chosen':
            return status
        return self.load_data_module.load_train_signals(filenames)


    def load_train_markups(self, filename):
        status = self.get_status_train_markups()
        if status == 'Markups have chosen':
            return status
        return self.load_data_module.load_train_markups(filename)


    def load_test_signal(self, filename):
        return self.load_data_module.load_test_signal(filename)


    def choose_model(self, path):
        try:
            model = self.load_data_module.load_model(path)
            if model:
                return self.classification_module.load_model(model)
```

Parameters

```

class Data:
    def __init__(self, status, data=None):
        self.data = data
        self.status = status
```

```

class AudioParams:

    @staticmethod
    def sr():
        return 16000

    @staticmethod
    def frame_sz():
        return 2500

    @staticmethod
    def hop_part():
        return 0.6

    @staticmethod
    def wavelet():
        return 'db4'

    @staticmethod
    def wavelet_level():
        return 6

    @staticmethod
    def min_phonem_interval():
        return 0.025
```

```
from sklearn.svm import SVC

class ClassificationParams:
    def __init__(self):
        self.model = SVC(kernel='rbf')
```

Parameters

```
1
2 class MarkupsParams:
3
4     @staticmethod
5     def filename():
6         return 'recordname'
7
8     @staticmethod
9     def intervals():
10        return 'intervals'
11
12    @staticmethod
13    def start_end():
14        return ['start', 'end']
```