

МИНОБРНАУКИ РОССИИ

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Казанский национальный исследовательский технологический университет»**

**(ФГБОУ ВО «КНИТУ»)**

Кафедра Интеллектуальных систем и управления информационными ресурсами

Направление 01.03.02

Специальность Прикладная математика и информатика

Группа 4391-22

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

Уровень образования Бакалавр

(бакалавр, специалист, магистр)

Вид ВКР Комбинированный

( проектный, исследовательский, комбинированный)

Тема Разработка голосового помощника для персонального компьютера\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Герасимов А.В.)

Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Антонова П.В.)

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Мангушева А.Р.)

Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Еремина О.В.)

2023 г.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc137233240)

[З А Д А Н И Е 4](#_Toc137233241)

[ЛИСТ НОРМОКОНТРОЛЕРА 5](#_Toc137233242)

[ГЛАВА I. ТЕОРЕТИЧЕСКИЕ И МЕТОДОЛОГИЧЕСКИЕ АСПЕКТЫ СОЗДАНИЯ ГОЛОСОВОГО ПОМОЩНИКА 6](#_Toc137233243)

[1.1 Обзор аналогов 6](#_Toc137233244)

[1.2 Обзор методов обработки естественного языка 8](#_Toc137233245)

[1.3 Постановка задачи 19](#_Toc137233246)

[ГЛАВА II. РАЗРАБОТКА ПРОЕКТА ГОЛОСОВОГО ПОМОЩНИКА 20](#_Toc137233247)

[2.1 Функциональные требования 20](#_Toc137233248)

[2.2 Архитектура голосового помощника 21](#_Toc137233249)

[2.3 Блок-схема голосового помощника 23](#_Toc137233250)

[ГЛАВА III. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ГОЛОСОВОГО ПОМОЩНИКА 26](#_Toc137233251)

[3.1 Разработка голосового помощника 26](#_Toc137233252)

[3.2. Тестирование голосового помощника 41](#_Toc137233253)

[ЗАКЛЮЧЕНИЕ 45](#_Toc137233254)

[СПИСОК ЛИТЕРАТУРЫ 46](#_Toc137233255)

# ВВЕДЕНИЕ

Сегодня неотъемлемой частью жизни современного человека стали голосовые помощники – сервисы, распознающие с помощью искусственного интеллекта человеческую речь и выполняющие озвученные голосом команды.

Подобные сервисы применяют в различном бизнесе в службах поддержки клиентов, так как их использование влечет за собой сокращение количества сотрудников центров обработки звонков, а, следовательно, и снижение затрат на их содержание. Также голосовые ассистенты активно используются в смартфонах, смарт колонках, некоторых браузерах с целью сделать управление программным обеспечением более удобным для пользователя.

Несмотря на популярность голосовых помощников в различных сферах, количество таких сервисов для управления персональным компьютером невелико: часть программ недоступна для российских пользователей, а большинство отечественных аналогов устарели и не поддерживаются своими разработчиками. Также существующим сервисам характерен важный недостаток: пользователю необходимо озвучивать команды, строго соответствующие заданным, так как ассистенты не способны воспринимать схожие формулировки. Данный недостаток делает использование сервиса менее удобным и говорит о неудовлетворительном качестве распознавания намерений.

Данная работа состоит из трех глав. В первой главе изложены теоретические и методологические аспекты создания голосового помощника: приведены обзор аналогов и обзор методов обработки естественного языка, а также поставлена задача. Во второй главе осуществляется разработка проекта системы: описываются функциональные требования, архитектура и блок-схема работы программы. В заключительной главе описаны разработка основных модулей программы и тестирование ее работы.

Кафедра ИСУИР «УТВЕРЖДАЮ»

Направление 01.03.02 Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Специальность Прикладная математика \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2023 г.

и информатика

Группа 4391-22

# З А Д А Н И Е

**на выпускную квалификационную работу студента** Ереминой Олеси Витальевны

Тема Разработка голосового помощника для персонального компьютера

Срок представления работы к защите «\_\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г.

Цель, задачи и исходные данные работы: Цель работы – сделать управление персональным компьютером для пользователя более удобным, предоставив возможность выполнять часть работы с помощью голосовых команд, в том числе, формулировки которых незначительно отличаются от заданных.

Задачи: распознавание и синтез речи, обработка естественного языка, реализация функционала, создание графического интерфейса.

Задание по разделам работы: Теоретические и методологические аспекты создания голосового помощника: обзор аналогов, обзор методов обработки естественного языка, постановка задачи;

Разработка проекта голосового помощника: функциональные требования, архитектура голосового помощника, блок-схема голосового помощника;

Разработка и тестирование голосового помощника: разработка голосового помощника, тестирование голосового помощника.

Содержание графической части (иллюстрированного материала): Презентационный материал

Консультанты: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Дата выдачи задания «\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 г.

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Мангушева А.Р. )

Задание принял к исполнению \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Еремина О.В. )

# ЛИСТ НОРМОКОНТРОЛЕРА

1. Лист является обязательным приложением к пояснительной записке дипломного (курсового) проекта.
2. Нормоконтролер имеет право возвращать документацию без рассмотрения в случаях:

-нарушения установленной комплектности,

-отсутствия обязательных подписей,

-нечеткого выполнения текстового и графического материала.

1. Устранение ошибок, указанных нормоконтролером, обязательно.

ПЕРЕЧЕНЬ

замечаний и предложений нормоконтролера по дипломному (курсовому) проекту, студента

4391-22, О.В. Ереминой

(группа, инициалы, фамилия)

|  |  |  |
| --- | --- | --- |
| Лист  (страница) | Условное обозна-  чение (код ошибок) | Содержание замечаний и предложений со ссылкой на нормативный документ, стандарт или типовую документацию |
|  |  |  |

Дата\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Нормоконтролер\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Антонова П.В.)

(подпись) (фамилия, инициалы)

# ГЛАВА I. ТЕОРЕТИЧЕСКИЕ И МЕТОДОЛОГИЧЕСКИЕ АСПЕКТЫ СОЗДАНИЯ ГОЛОСОВОГО ПОМОЩНИКА

# Обзор аналогов

В данном разделе рассмотрены одни из самых популярных современных голосовых помощников.

Яндекс Алиса – виртуальный голосовой ассистент от компании «Яндекс». Алиса способна распознавать естественную речь и имитировать живой диалог на русском языке, отвечая на вопросы пользователя. Работает в приложении «Яндекс», а также в «Яндекс.Браузере», поддерживается на платформах Android и iOS и на персональных компьютерах. Алиса имеет широкий функционал. Например, она помогает в управлении умным домом, получении информации о погоде, прослушивании музыки и взаимодействии с другими сервисами от компании «Яндекс» [1].

Ассистент Google – ассистент от «Google», используемый в смартфонах и приложении «Google Allo», а также в продуктах компании: умных часах и домах. Данный помощник поддерживает русский язык, способен помочь в решении множества задач, особенно в работе с сервисами от «Google» [2].

Маруся – помощник от компании «VK», доступный в приложениях на смартфонах, в почте «Mail.ru» и во «Вконтакте». Также Маруся работает в умных колонках, умных часах, ТВ-приставках и умных домах [3].

Amazon Alexa – помощник от компании «Amazon», работающий на смартфонах и умных колонках. Отличительной чертой данного сервиса является возможность расширения возможностей с помощью установки пакетов от сторонних поставщиков [4].

Олег – виртуальный голосовой ассистент от «Тинькофф», доступный в мобильном приложении банка и специализирующийся на финансовых вопросах [5].

Салют от «Сбера» отличается возможностью выбирать понравившегося ассистента, включая в себя сразу три помощника с разными голосами. Он работает в мобильных приложениях банка и интеллектуальных устройствах «Сбера» и сторонних производителей [6].

Рассмотренные выше сервисы поддерживают русский язык и характеризуются хорошим качеством распознавания и синтеза речи, высокой точностью классификации намерений и широким перечнем решаемых задач. Популярность голосовых ассистентов растет с каждым годом, их используют в различных сферах, тем не менее, количество ассистентов для персональных компьютеров на данный момент не большое.

Cortana– официальный голосовой помощник для Windows от «Microsoft», интегрированный в операционную систему и способный управлять ей и приложениями во время работы. Однако имеет важный недостаток для российского пользователя: Cortana не поддерживает русский язык [7].

Siri – облачный персональный ассистент, поддерживаемый исключительно на устройствах компании «Apple» [8].

Большинство отечественных аналогов, такие как «Typle» и «Горыныч», были разработаны в конце девяностых годов прошлого или начале нулевых годов нашего века, их функционал скуден, а разработчики давно не поддерживают данные продукты.

Laitis– программа для голосового управления персональным компьютером с операционной системой Windows. От других отечественных аналогов отличается широким функционалом, но имеет важные недостатки: она является скорее обработчиком пользовательских команд, чем полноценным помощником, а также имеет низкое качество распознавания команд. Программа «Laitis» способна реагировать лишь на команды, строго соответствующие заранее описанным, не воспринимая схожие формулировки, что неудобно для пользователя, которому необходимо помнить команды наизусть.

Таким образом, существующие сегодня голосовые помощники для управления персональным компьютером имеют ряд недостатков. Главный из них - неудовлетворительное решение задачи классификации намерений пользователя, очень важной задачи всех голосовых помощников и чат-ботов [9].

# Обзор методов обработки естественного языка

Обработка естественного языка — это пересечение искусственного интеллекта и математической лингвистики, изучающее методы анализа и синтеза естественного языка и включающее в себя три направления: распознавание речи, понимание и генерация естественного языка [10].

* + 1. Распознавание и синтез речи

Распознавание речи – это автоматический процесс, представляющий собой преобразование звука в цифровую информацию. Одним из видов распознавания речи является технология «speech-to-text», заключающаяся в переводе речевого сигнала в текстовые данные. Основой работы «speech-to-text» являются нейронные сети [11].

Синтез речи – это процесс генерации человеческой речи. Технология «text-to-speech» подразумевает генерацию на основе текста [12].

Сегодня распознавание и синтез речи являются актуальными задачами, поэтому количество сервисов для их решения растет. В данном разделе рассматриваются несколько популярных сервисов, позволяющих использовать «speech-to-text» и «text-to-speech» технологии:

1. Speech Recognition – библиотека для использования нескольких сервисов распознавания речи: CMU Sphinx, Google Cloud Speech API, Wit.ai, распознавание голоса от Microsoft Bing, IBM Speech Text, Houndify API и др.
2. Vosk – библиотека с открытым исходным кодом, поддерживающая распознавание множества языков, в том числе и русского.
3. Yandex SpeechKit – сервис от компании «Яндекс», позволяющий как распознавать, так и синтезировать речь.
4. Google cloud speech и text to speech – сервисы от «Google» для распознавания и синтеза.
5. Microsoft Speech – сервис от компании «Microsoft» также поддерживающий как синтез, так и распознавание.
6. Pyttsx3 – библиотека языка Python для синтеза речи в режиме офлайн.
7. Silero – система для синтеза речи, имеющая высокую «человечность» голосов.

Большинство сервисов для распознавания и синтеза речи требуют подключения к интернету, из-за чего процесс существенно затормаживается в случае нестабильного и медленного соединения.

В данной работе для распознавания речи используется библиотека для python Vosk, преимуществами которой являются: возможность распознавать речь офлайн, небольшой вес модели (87 мб) и скорость распознавания. Для синтеза речи – библиотека pyttsx3, которая также работает быстро и офлайн.

* + 1. Предобработка текста

Алгоритмы машинного обучения не способны работать с текстами на естественном языке, поэтому необходима предварительная обработка текста. С помощью предобработки текст на естественном языке приводится к формату, удобному для дальнейшей работы. Этот процесс может состоять из нескольких этапов, отличающихся в зависимости от поставленной задачи и её реализации.

Как правило, первым этапом обработки текста является нормализация.

Нормализация – это операция, в процессе которой текст приводится к нужному регистру, в нём удаляются знаки пунктуации, числа и лишние пробельные символы.

Следующим шагом является токенизация.

Токенизация заключается в разбиении текста на слова или словосочетания - токены.

Важным, но необязательным шагом предобработки является удаление стоп-слов, не несущих смысловой нагрузки: союзов, предлогов и других неинформативных слов, и шаблонов.

Далее необходимо провести нормализацию слова. Одним из способов является стемминг.

Велико количество слов, значения которых схожи, но написание отличается суффиксами, окончаниями, приставками и другими частями слова. Этот факт существенно усложняет создание словарей и дальнейшую обработку. Для того чтобы привести слово к основной форме, с его конца и начала последовательно отрезают части. Правила стемминга создаются заранее и обычно представляют собой регулярные выражения.

Альтернативным способом нормализации слова является лемматизация. Идея данного подхода заключается в приведении слова к его словарной форме:

* существительные приводятся к именительному падежу и единственному числу;
* прилагательные — именительному падежу, единственному числу и мужскому роду;
* глаголы, причастия и деепричастия заменяются глаголом в инфинитиве несовершенного вида.

Отличие методов нормализации слова в том, что стемминг не учитывает контекст и разницу между словами, имеющими разный смысл, но схожее написание [13].

* + 1. Векторизация текста

Следующим этапом работы после предобработки текста является процесс векторизации, заключающийся в сопоставлении слов с пространством векторов. Для конвертации текста в векторы существует четыре основных подхода: прямое кодирование, мешок слов, TF-IDF векторизация и word **e**mbeddings.

Прямое кодирование (one-hot encoding) – способ преобразования токенов (слов) в тензоры, при котором сначала каждый токен представляется бинарным вектором, затем элементу, соответствующему номеру токена в словаре, ставится единица. Пример прямого кодирования слов представлен на рисунке 1.

Большая размерность векторного представления документа является основной проблемой прямого кодирования. С увеличением количества слов в словаре увеличивается матрица.

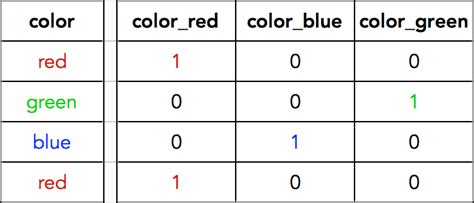


Рисунок 1 – Пример прямого кодирования слов.

Мешок слов (Bag of words). В данной модели текст представляется в виде вектора – мультимножества его слов без учета грамматики, порядка слов, но с учетом информации о количестве слов. На рисунке 2 представлен пример кодирования предложений с помощью данного метода.

«Мешок слов» решает проблему размерности по одной оси. Однако данный метод не учитывает важность токенов на основе их встречаемости.

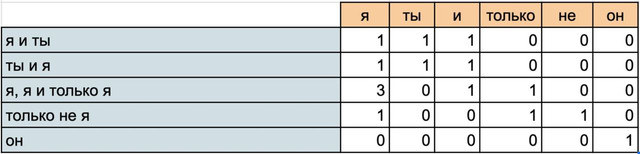


Рисунок 2 – Пример кодирования текста «Мешком слов».

TF-IDF- статистический показатель, применяемый для оценки важности слова в контексте категории, документа или коллекции документов. На рисунке 3 представлена наглядная иллюстрация TF-IDF векторизации.

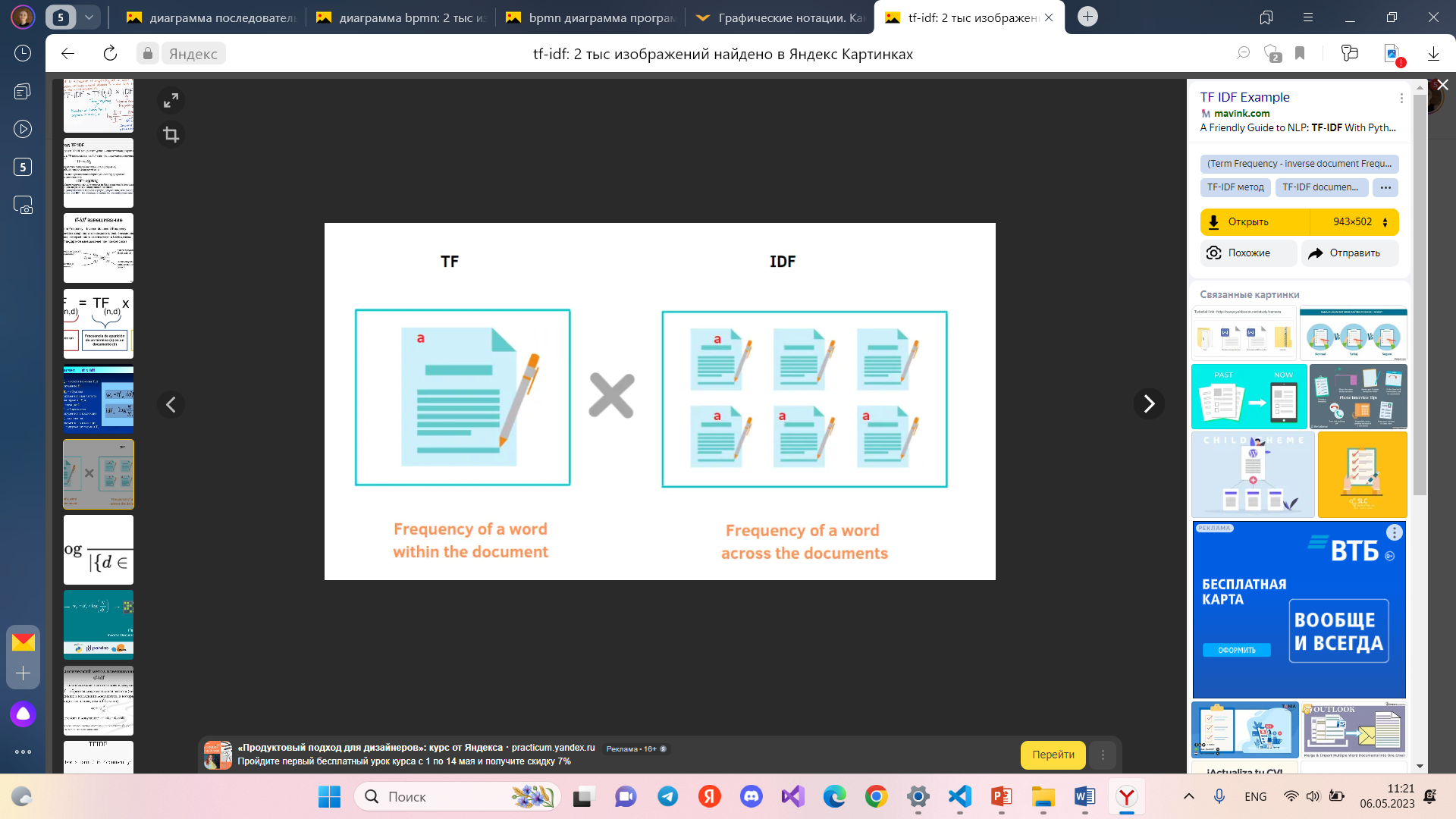


Рисунок 3 – Иллюстрация TF-IDF векторизации.

TF-IDF состоит из двух компонентов: Term Frequency (частотность слова в документе) и Inverse Document Frequency (инверсия частоты документа). Расчет значений производится в соответствии с формулами (1) и (2).

где  — сколько раз встречается токен в i-ом документе,  
 — общее количество токенов в -ом документе.

где  — количество документов, в которых встречается токен,  
 — общее количеств документов.

В конечном счете, согласно формуле (3), [TF-IDF](https://python-school.ru/wiki/tf-idf/) – это произведение TF на IDF.

TF считается для токенов документа, тогда как IDF – токенов всего корпуса [14].

Word embeddings.Embedding – англоязычный термин, употребляемый из-за отсутствия подходящего термина в русском языке, обозначающий сопоставление произвольной сущности некоторому вектору. Между разными словами можно выделить два вида близости: лексическая близость и семантическая. Embedding используется для представления слова в виде вектора, отображающего его значение в некотором «пространстве смыслов».

На рисунке 4 представлен пример расположения векторов в натренированной модели «word2vec».

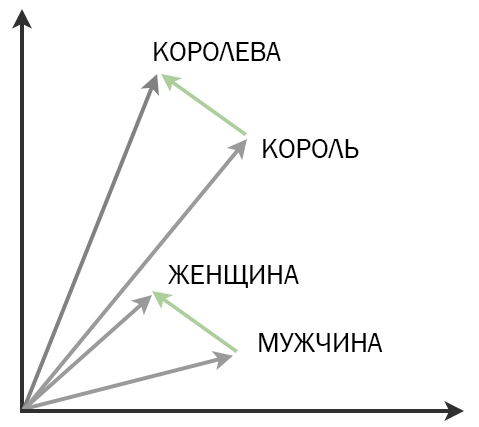


Рисунок 4 – Пример расположения векторов в натренированной модели «word2vec».

Слово "король" относится к слову "королева" так же, как слово "мужчина" к слову "женщина", что для нас совершенно естественно и понятно, но в других моделям добиться такого же соотношения векторов можно только с помощью специальных ухищрений. Здесь же — это происходит естественно из самого корпуса текстов. Помимо семантических связей, улавливаются и синтаксические, справа показано соотношение единственного и множественного числа.

В данной работе используется TF-IDF векторизация, поскольку данный метод учитывает встречаемость слова как в одном документе, так и во всех документах, относительно прост в реализации и не требует большого количества данных для обучения [15].

* + 1. Классификация намерений пользователя

Задача классификации – это задача, в которой существует множество объектов, разделенных на классы. Имеется конечное множество объектов, для которых известно, к каким классам они принадлежат. Такое множество называют выборкой. Для других объектов классовая принадлежность неизвестна. Требуется создать алгоритм, позволяющий классифицировать произвольный объект из исходного множества, т.е. указать номер или наименование класса, к которому данный объект относится.

Классификация намеренийподразумевает выявление намерения пользователя по введенному тексу.

В качестве обучающей выборки используется таблица, содержащая примеры входного текста и соответствующие им намерения.

В первую очередь входной текст на естественном языке следует предварительно обработать для повышения качества работы алгоритма классификации.

Затем необходимо векторизовать текст (представить в виде набора чисел) для последующей подачи в алгоритм.

Заключительный этап представляет собой обучение модели и непосредственно предсказание намерений по входным данным.

Scikit-learn– одна из самых популярных библиотек для выполнения алгоритмов классификации, регрессии и кластеризации на Python. Она дает доступ ко множеству различных алгоритмов. В данном разделе представлен обзор основных алгоритмов классификации этой библиотеки.

Метод k-ближайших соседей (K-Nearest Neighbors) заключается в поиске кратчайшей дистанции между тестируемым объектом и ближайшими к нему классифицированными объектами из обучающей выборки. Классифицируемый объект будет относиться к классу, к которому принадлежит ближайший объект выборки. Иллюстрация данного метода представлена на рисунке 5.

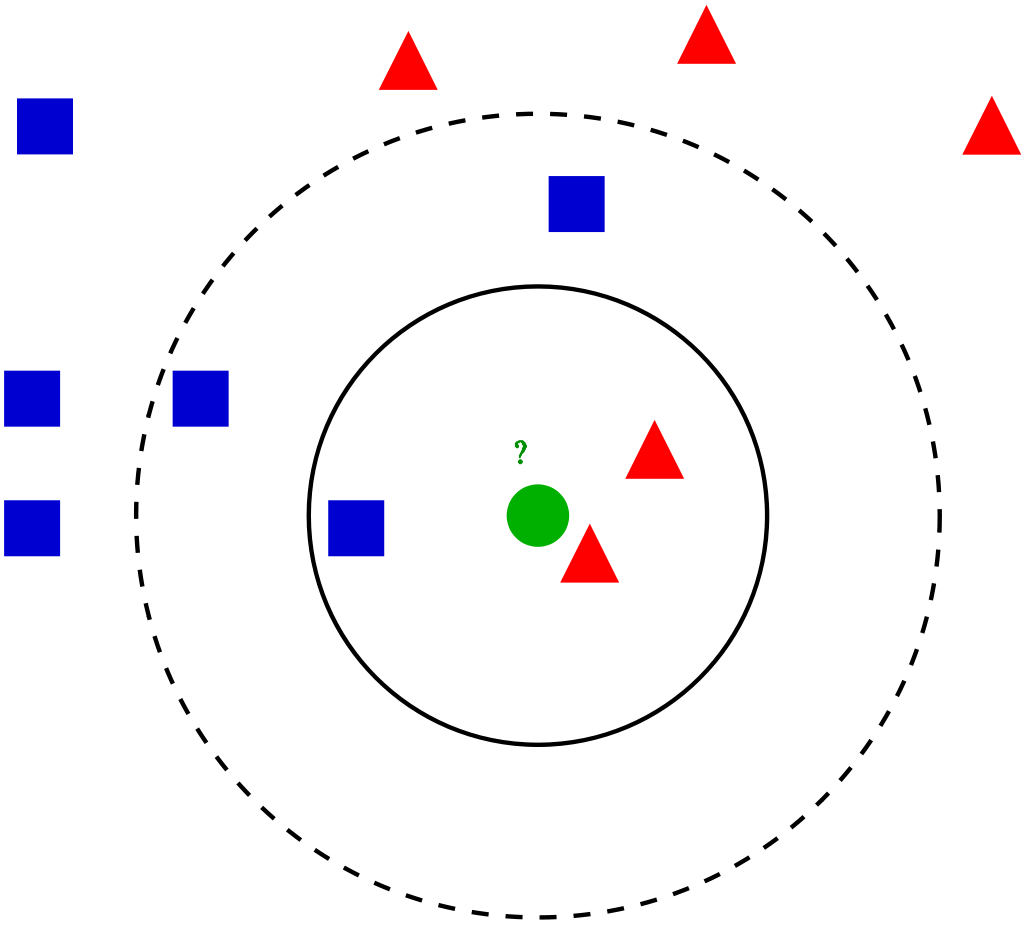


Рисунок 5 – Визуальное представление метода k-ближайших соседей.

Классификатор дерева решений (Decision Tree Classifier) разбивает данные на всё меньшие и меньшие подмножества на основе разных критериев, т.е. у каждого подмножества своя сортирующая категория. С каждым разделением количество объектов определенного критерия уменьшается. Классификация завершится, когда сеть дойдет до подмножества только с одним объектом. При объединении нескольких подобных деревьев решений, получается Случайный Лес (Random Forest).

Иллюстрация классификатора дерева решений представлена на рисунке 6.

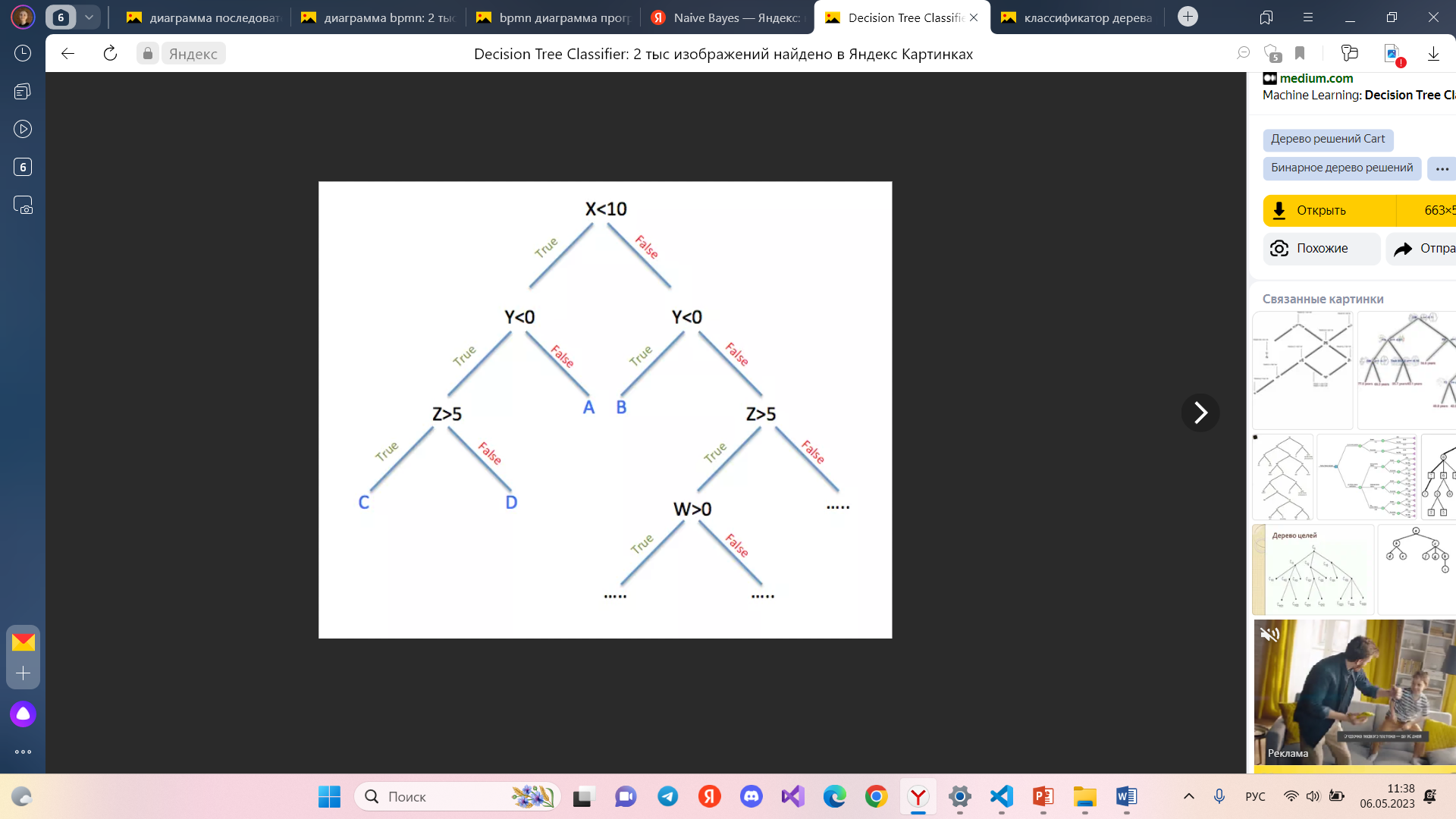


Рисунок 6 – Визуальное представление классификатора дерева решений.

Наивный байесовский классификатор (Naive Bayes) вычисляет вероятность принадлежности объекта к какому-либо классу. Вероятность вычисляется из шанса, что какое-то событие произойдет, с опорой на уже произошедшие события. На рисунке 7 представлена иллюстрация данного метода.

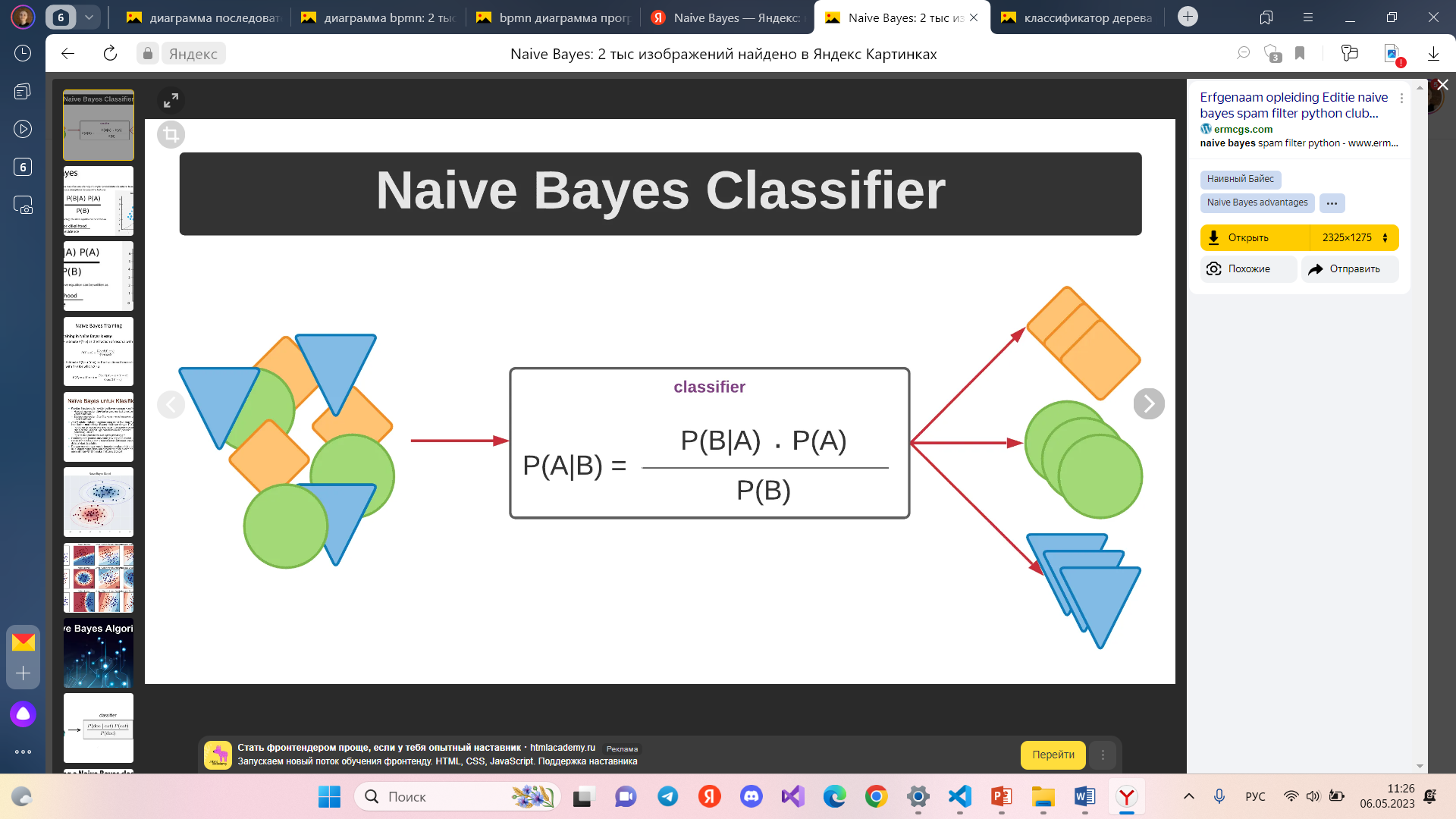


Рисунок 7 – Иллюстрация наивного байесовского классификатора.

Линейный дискриминантный анализ (Linear Discriminant Analysis) работает путем уменьшения размерности набора данных, проецируя все точки данных на линию. Затем он комбинирует точки в классы, базируясь на их расстоянии от центральной точки. Данный метод относится к линейным алгоритмам классификации и подходит для данных с линейной зависимостью. Иллюстрация приведена на рисунке 8.

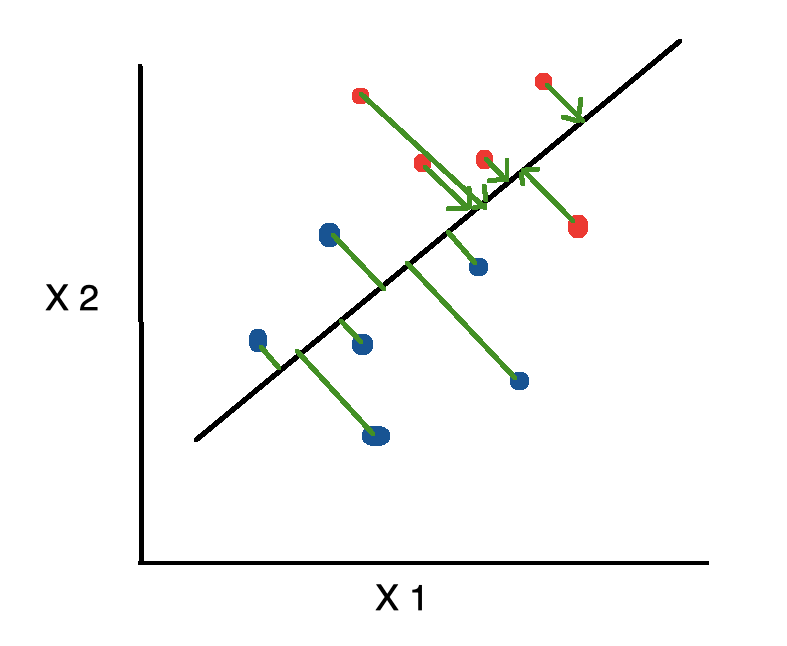


Рисунок 8 – Визуальное представление линейного дискриминантного анализа.

Метод опорных векторов (Support Vector Machines) заключается в рисовании линии между разными кластерами точек, которые нужно сгруппировать в классы. С одной стороны линии будут точки, принадлежащие одному классу, с другой – к другому. Классификатор будет пытаться увеличить расстояние между рисуемыми линиями и точками на разных сторонах, чтобы увеличить свою «уверенность» определения класса. Когда все точки построены, сторона, на которую они падают – это класс, которому эти точки принадлежат.

Визуальное представление работы метода опорных векторов приведено на рисунке 9.

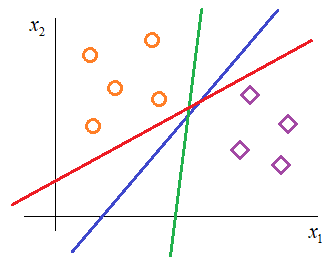


Рисунок 9 – Визуальное представление метода опорных векторов.

Логистическая регрессия (Logistic Regression) - статическая модель, используемая для прогнозирования вероятности возникновения некоторого события путём подгонки данных к логистической кривой. Иллюстрация метода представлена на рисунке 10.

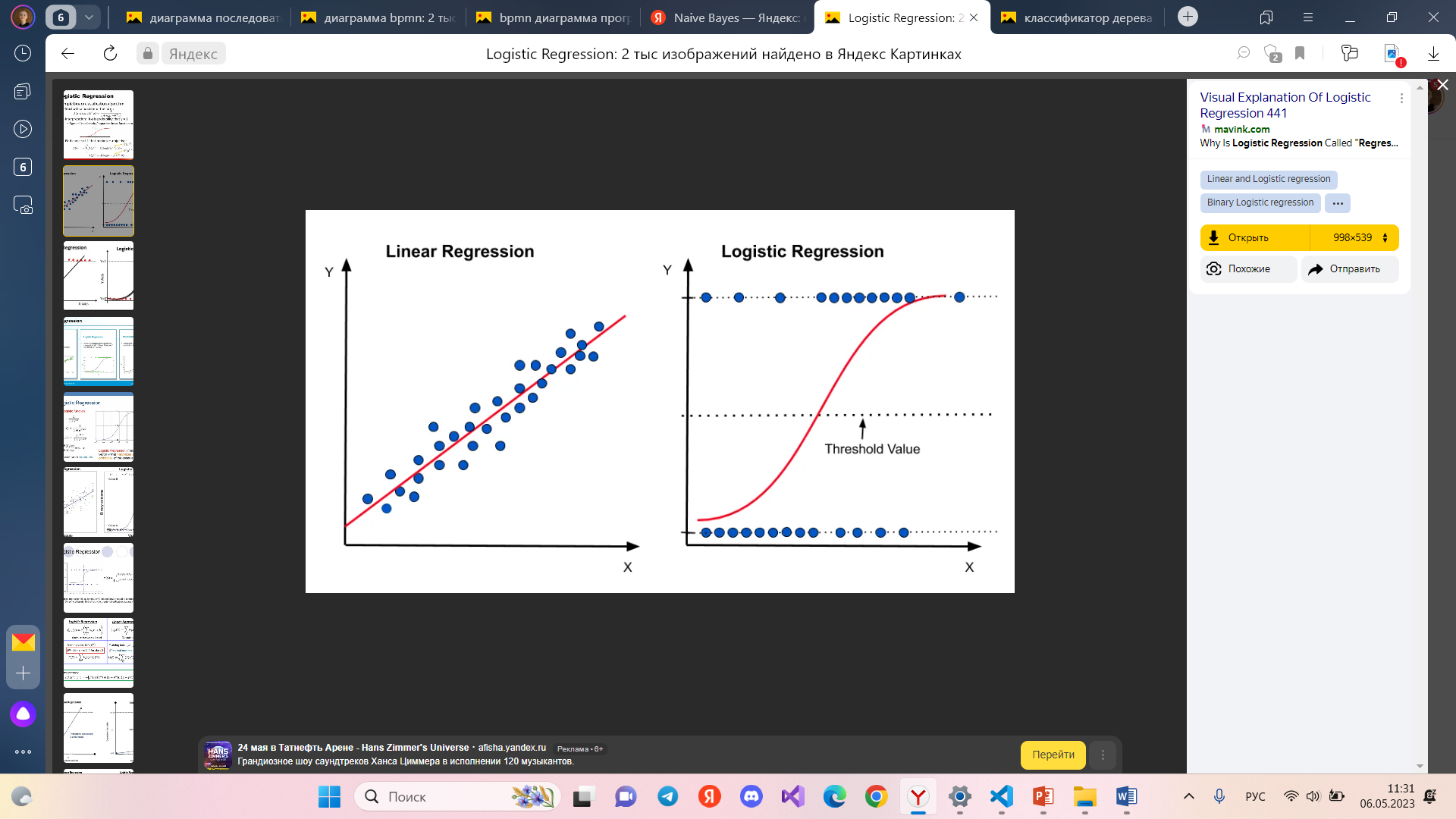


Рисунок 10 – Визуальное представление логистической регрессии.

В классическом виде процесс машинного обучения состоит из предварительной обработки данных, извлечения, выбора и построения признаков, выбора модели машинного обучения, оптимизации гиперпараметров и оценки полученной модели. Осуществление всех этапов вручную – это объемная и трудоемкая задача. В решении этой проблемы на помощь специалистам по машинному обучению приходит автоматическое машинное обучение [16].

Автоматическое машинное обучение (AutoML) – это процесс создания динамической комбинации различных методов для формирования простой в использовании сквозной конвейерной системы машинного обучения. AutoML используется как для автоматизации всех этапов обучения, так и для автоматизации части из них.

В данной работе подбор оптимальной модели и гиперпараметров осуществляется с помощью библиотеки auto-sklearn.

Auto-sklearn– это автоматизированный пакет машинного обучения, позволяющий осуществлять автоматический выбор наиболее подходящего алгоритма из библиотеки scikit-learn и настройку гиперпараметров [17].

# Постановка задачи

Цель данной работы – сделать управление персональным компьютером для пользователя более удобным, предоставив возможность выполнять часть работы с помощью голосовых команд, в том числе, формулировки которых незначительно отличаются от заданных.

Основными задачами являются:

1. Распознавание и синтез речи
2. Обработка естественного языка
3. Реализация функционала
4. Создание графического интерфейса

Программа должна обрабатывать голосовые команды пользователя, определять его намерение, отвечать и выполнять озвученную команду.

# ГЛАВА II. РАЗРАБОТКА ПРОЕКТА ГОЛОСОВОГО ПОМОЩНИКА

# Функциональные требования

Функциональные требования к системе:

1. При нажатии пользователем на кнопку «Пуск» запускается запись и обработка речи;
2. При нажатии пользователем на кнопку «Стоп» останавливается запись и обработка речи;
3. При озвучивании пользователем команды вместе со словом-триггером, выполняется соответствующая функция:
4. Если пользователь озвучил намерение узнать погоду, выполняется http-запрос, обрабатывается полученный ответ, и помощник озвучивает погоду;
5. Если пользователь озвучил намерение узнать время, помощник сообщает время;
6. Если пользователь озвучил намерение открыть какую-либо программу, запускается исполняемый файл озвученной программы;
7. Если пользователь озвучил намерение открыть веб-сайт, открывается новая вкладка в браузере с соответствующим сайтом;
8. Если пользователь озвучил намерение выключить помощника, останавливается запись и обработка речи;
9. При нажатии пользователем на кнопку сохранения настроек, внесенные изменения сохраняются в json-файл.

На рисунке 11 представлена диаграмма вариантов использования системы.

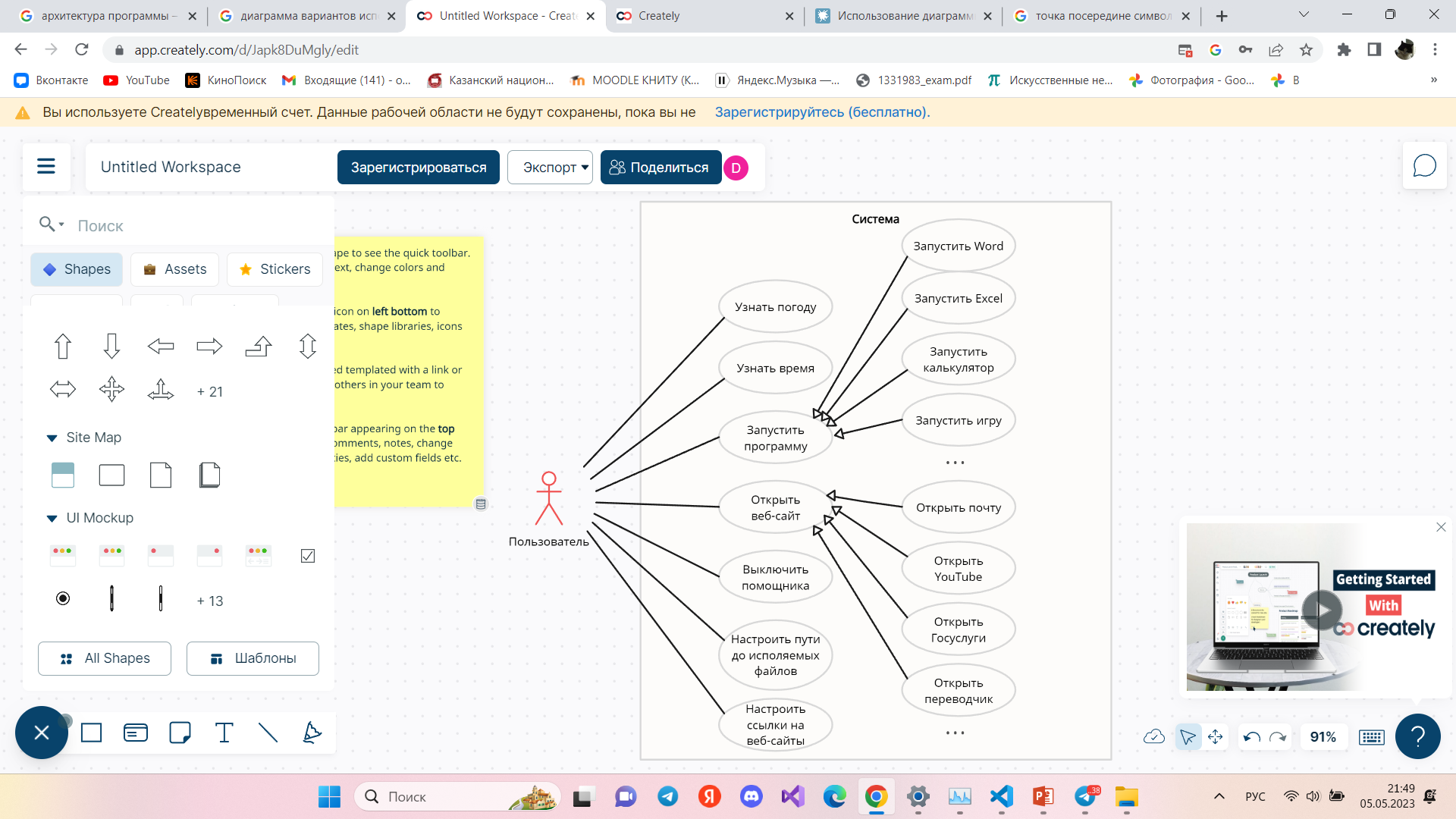


Рисунок 11 – Диаграмма вариантов использования.

# Архитектура голосового помощника

На рисунке 12 представлена архитектура системы, согласно которой пользователь, взаимодействуя с графическим интерфейсом, может инициировать запуск и остановку работы модуля обработки речи и вносить изменения в настройки с помощью модуля настроек путей и ссылок. Модуль настроек считывает информацию из json-файлов с путями и ссылками, а также вносит в них изменения.

При запуске модуля обработки речи, пользователь может взаимодействовать с ним непосредственно, озвучивая команды голосом, а также с модулем синтеза и озвучки речи, получая аудио-ответ.

Модуль обработки речи загружает данные для обучения модели из json-файлов, а также имеет доступ к перечню намерений пользователя и формулировок ответов для вызова функции, соответствующей озвученной команде, и инициирования синтеза и озвучки необходимого ответа.

Модуль функций получает ссылки на веб-сайты и пути до исполняемых файлов из модуля настроек путей и ссылок и может также инициировать синтез и озвучку ответа для пользователя.

Модуль синтеза и озвучки также выполняет функцию вывода ответа на графический интерфейс в виде текста.

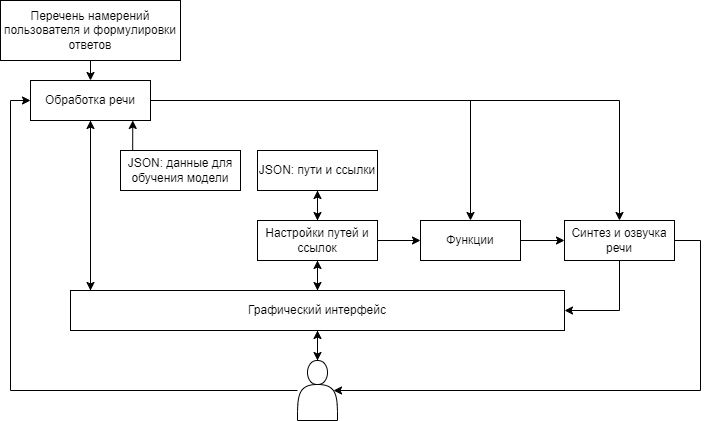


Рисунок 12 – Архитектура системы.

# Блок-схема голосового помощника

На рисунке 13 представлена блок-схема работы системы.

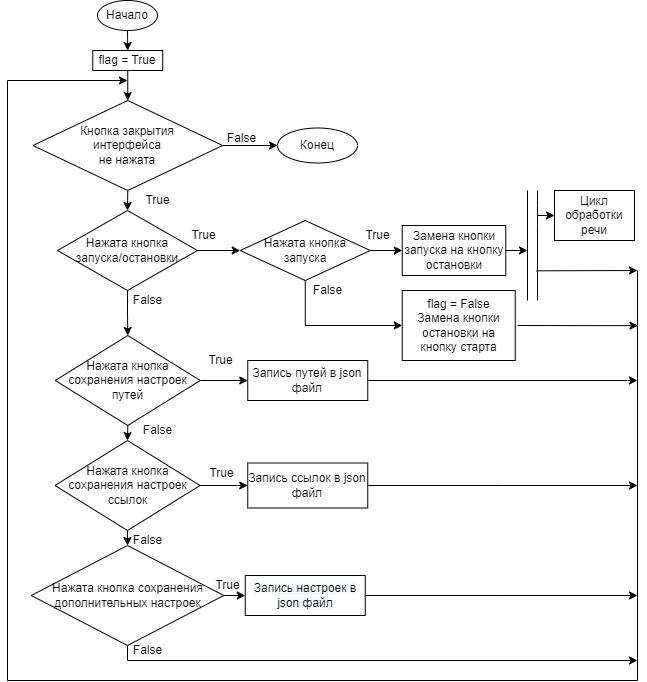


Рисунок 13 – Блок-схема программы.

После начала работы системы устанавливается значение флага, равное «True». Флаг необходим для прерывания работы параллельного процесса.

При нажатии пользователем одной из кнопок сохранения настроек, происходит запись изменений в json-файл.

При нажатии на кнопку запуска, происходит замена кнопки запуска на кнопку остановки и запускается цикл обработки речи в параллельном потоке. Параллельный поток необходим для корректной работы графического интерфейса.

При нажатии на кнопку остановки, устанавливается значение флага «False» и кнопка остановки заменяется кнопкой запуска.

Завершение работы происходит при нажатии кнопки закрытия интерфейса.

На рисунке 14 представлена блок-схема подпрограммы, запущенной в параллельном потоке.

После начала работы происходит обучение модели, после чего запускается цикл обработки речи, который работает, пока значение флага равно «True», и завершается при изменении значения флага на «False» в родительском потоке.

В теле цикла происходит запись речи. Запись продолжается до окончания фразы, после чего записанная речь распознается в текст и лемматизируется. Если в полученной текстовой фразе содержится слово-триггер, происходит его удаление, векторизация текста и классификация намерения пользователя. Затем выполняется функция, соответствующая классифицированному намерению, синтезируется и озвучивается ответ. Также завершение работы цикла может произойти в случае выполнения функции отключения помощника.

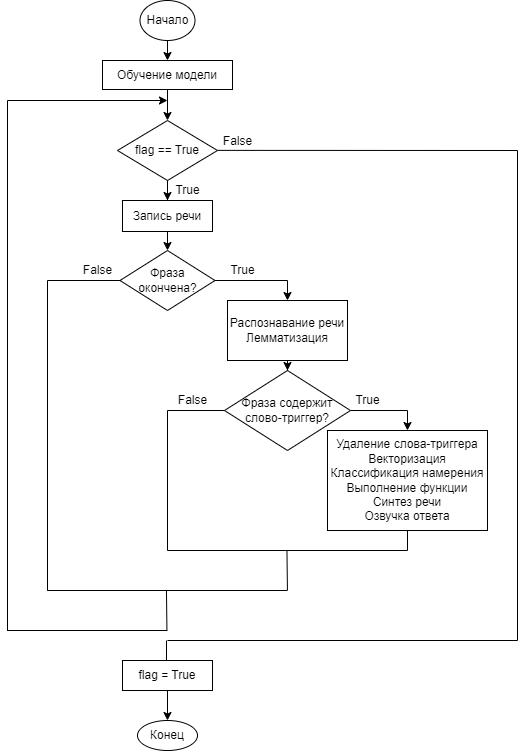


Рисунок 14 – Блок-схема подпрограммы.

# ГЛАВА III. РАЗРАБОТКА И ТЕСТИРОВАНИЕ ГОЛОСОВОГО ПОМОЩНИКА

# Разработка голосового помощника

* + 1. Хранение информации о намерениях пользователя

Намерения пользователя хранятся в файле «words.py», который содержит:

* «Триггеры» (слова-позывные ассистента) и созвучные с ними слова:

TRIGGERS = {'зефир', 'сфер', 'эфир', 'зефира', 'зверь', 'сефер'}

* Списки функций, приложений и веб-сайтов, а также список функций, инициирующих синтез и озвучку речи:

FUNCS={'offBot','get\_weather', 'time'}

APPS={'telegram', 'calc', 'game', 'excel', 'paint', 'powerpoint', 'terminal', 'word', 'settings', 'calendar', 'notebook', 'photo', 'camera', 'watch'}

WEB\_SITES={'browser', 'mail', 'youtube', 'vk', 'kinopoisk', 'yandexmusic', 'wikipedia', 'ok', 'google', 'rambler', 'avito', 'gismeteo', 'ozon', 'rbc', 'yandexmarket', 'gosuslugi', 'yandexeda', 'googletranslate'}

SPEAKERS={'get\_weather', 'time'}

* Словарь намерений, содержащий для каждого намерения примеры входного текста и варианты ответов:

INTENTS = {

"hello": {

"examples": ["привет", "приветик", "приветики", "здравствуйте", "здравствуй", "приветствую", "доброе утро", "добрый день", "добрый вечер"],

"responses": ["привет", "здравствуйте"],

},

…}

* + 1. Предобработка данных

Предобработка данных реализована в виде скрипта **«**lemmatization.py**»**, который выполняется на этапе подготовки данных и не используется непосредственно в работе программы.

Предобработка осуществляется посредством лемматизации с использованием библиотеки «Mystem» от компании «Яндекс».

Импортирование модулей:

from pymystem3 import Mystem #лемматизатор

import json #модуль для загрузки и сохранения файлов

import words #модуль с данными

Инициализация лемматизатора:

myst = Mystem()

Функция лемматизации текста:

def diction\_form(text):

text = ''.join(myst.lemmatize(text)).rstrip('\n')

return text

Создание массивов для обучения и заполнение их обработанными примерами входного текста и намерениями:

x = []

y = []

for intent in words.INTENTS:

examples = words.INTENTS[intent]["examples"]

for example in examples:

x.append(diction\_form(example)) #лемматизация каждого примера

y.append(intent)

Сохранение массивов в формате json:

with open('json/x.json', 'w') as filehandle:

json.dump(x, filehandle)

with open('json/y.json', 'w') as filehandle:

json.dump(y, filehandle)

* + 1. Подбор модели и гиперпараметров машинного обучения

Подбор модели и гиперпараметров машинного обучения реализован с использованием библиотеки auto-sklearn в облачной среде «Google Colab».

Загрузка и импортирование модулей:

!pip install auto-sklearn

!pip install numpy --upgrade

import autosklearn.classification #библиотека автоматического машинного обучения для задачи классификации

import json #модуль для загрузки и сохранения файлов

from sklearn.feature\_extraction.text import TfidfVectorizer #библиотека для векторизации текста

from sklearn.metrics import classification\_report #модуль для оценки метрик

Загрузка данных для обучения модели:

x = []

y = []

with open('x.json', 'r') as filehandle:

x=json.load(filehandle)

with open('y.json', 'r') as filehandle:

y=json.load(filehandle)

Загрузка тестовых данных, состоящих из примеров, не включенных тренировочный набор:

x\_test = []

y\_test = []

with open('x\_test.json', 'r') as filehandle:

x\_test=json.load(filehandle)

with open('y\_test.json', 'r') as filehandle:

y\_test=json.load(filehandle)

Векторизация данных:

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit\_transform(x)

vectors\_test = vectorizer.transform(x\_test)

Инициализация и подбор модели:

autosklearn\_classifier = autosklearn.classification.AutoSklearnClassifier(

time\_left\_for\_this\_task=120,

ensemble\_kwargs={"ensemble\_size": 1},)

autosklearn\_classifier.fit(vectors, y)

Вывод результата подбора продемонстрирован на рисунке 15:

print(autosklearn\_classifier.leaderboard())

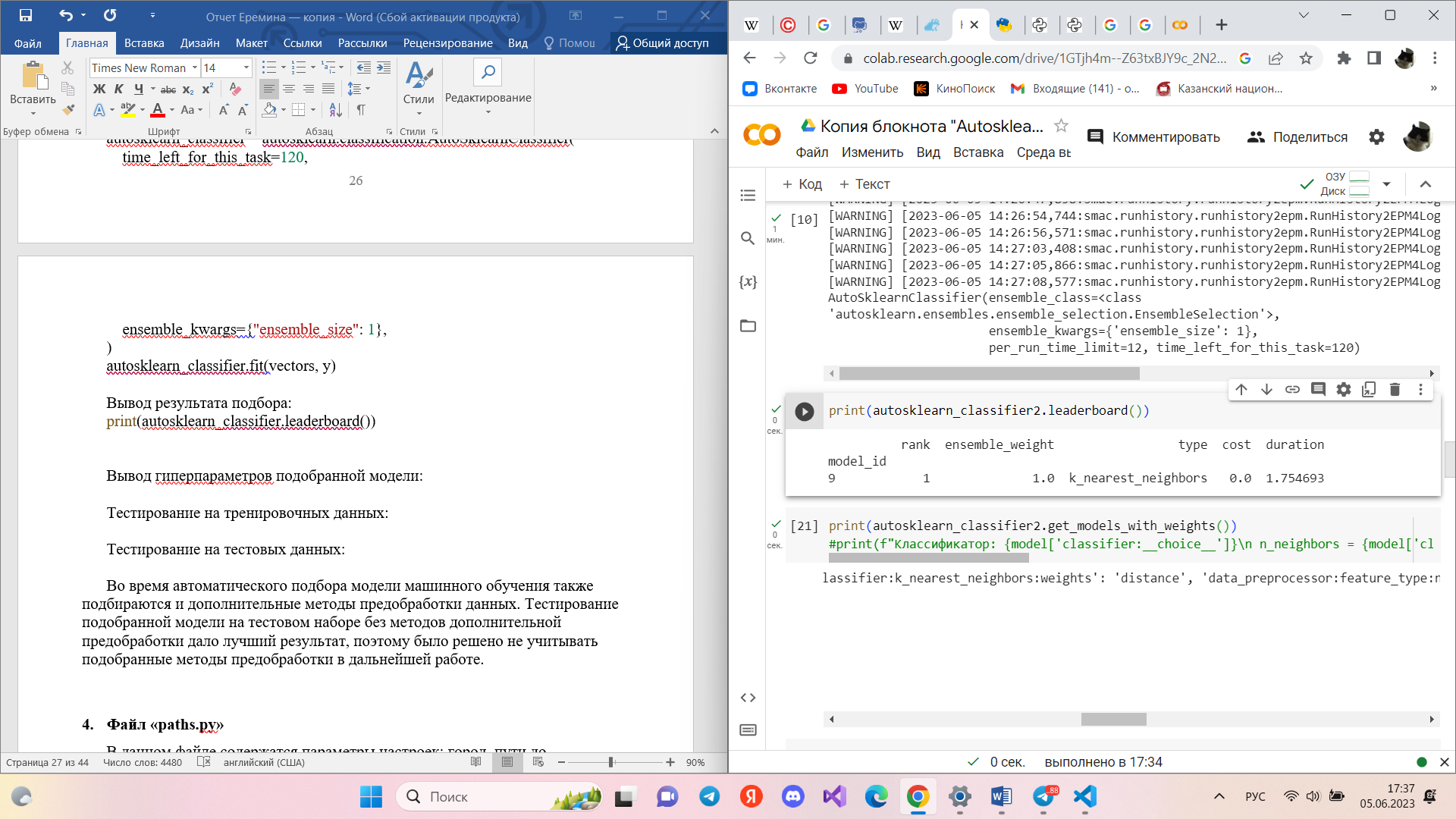


Рисунок 15 – Результат автоматического подбора модели машинного обучения.

Вывод гиперпараметров подобранной модели представлен на рисунке 16:

print(autosklearn\_classifier.get\_models\_with\_weights())

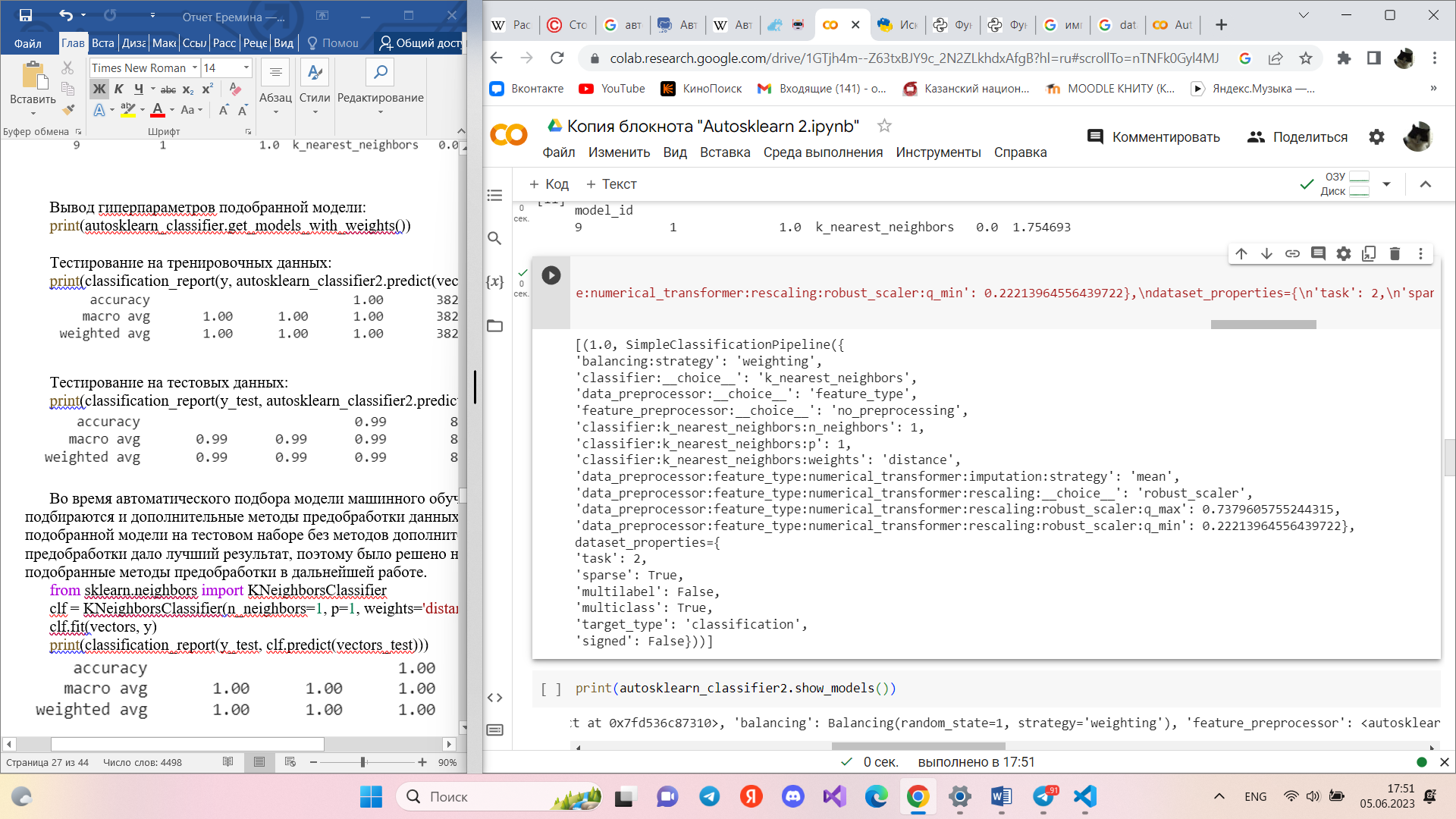


Рисунок 16 – Вывод гиперпараметров модели.

Оценка подобранной модели на тренировочных данных продемонстрирована на рисунке 17:

print(classification\_report(y, autosklearn\_classifier2.predict(vectors)))

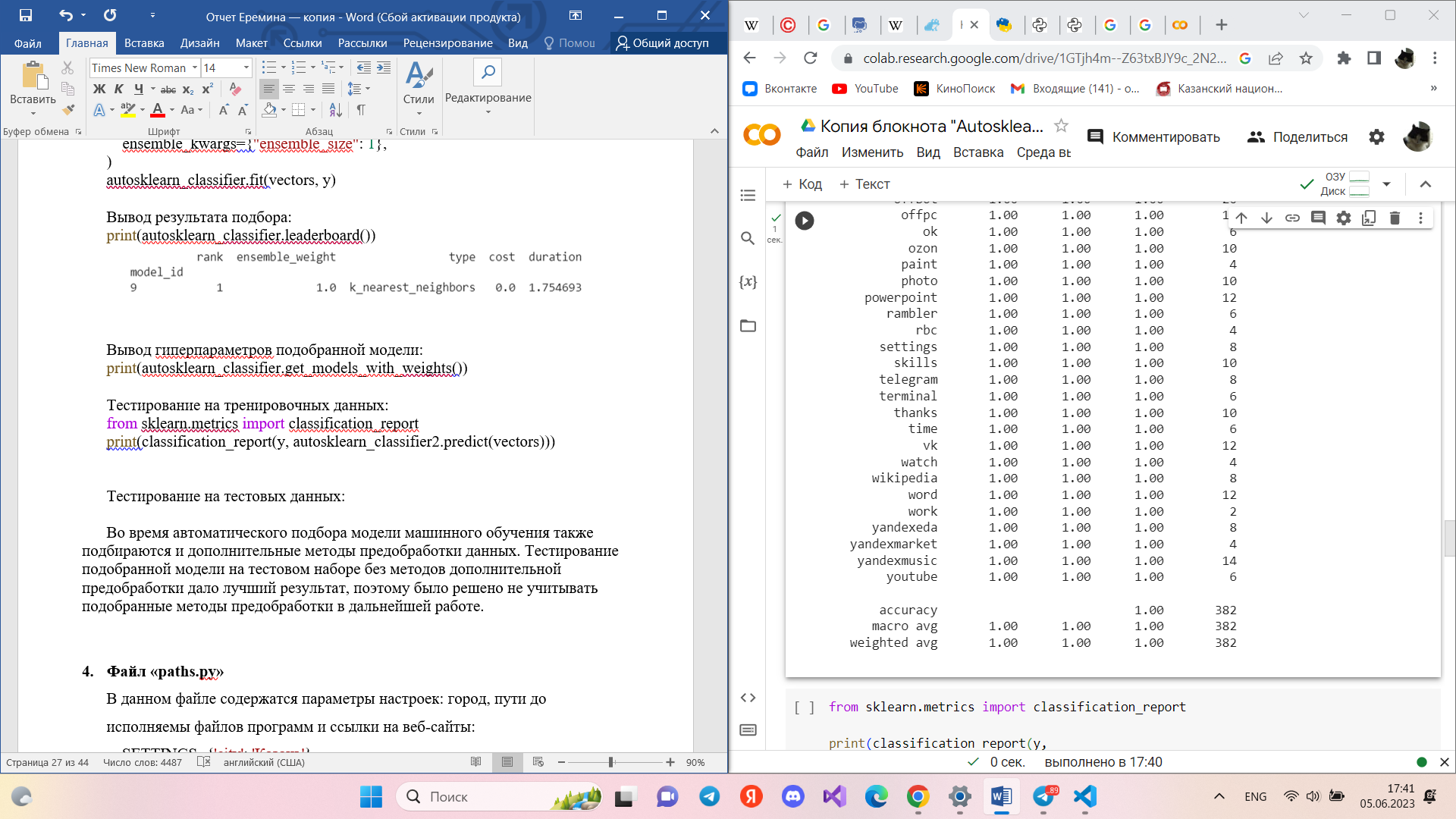


Рисунок 17 – Оценка модели на тренировочных данных.

Оценка подобранной модели на тестовых данных продемонстрирована на рисунке 18:

print(classification\_report(y\_test, autosklearn\_classifier2.predict(vectors\_test)))

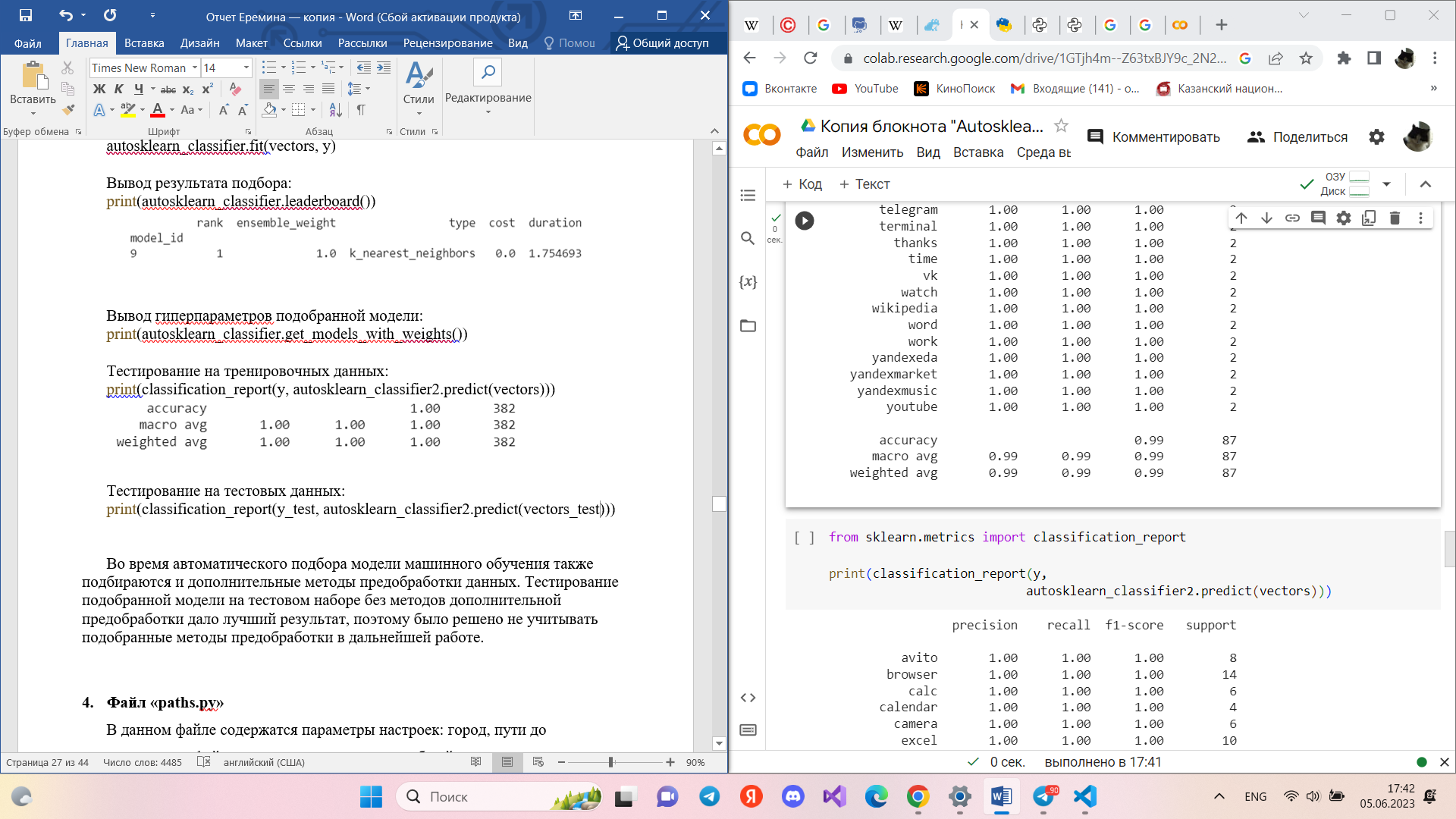


Рисунок 18 – Оценка модели на тестовых данных.

Во время автоматического подбора модели машинного обучения также подбираются и дополнительные параметры предобработки данных. Тестирование подобранной модели на тестовом наборе без дополнительной предобработки дало лучший результат, поэтому было решено не учитывать подобранные параметры предобработки в дальнейшей работе. Оценка подобранной модели на тестовых данных без учета предобработки продемонстрирована на рисунке 19:

from sklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n\_neighbors=1, p=1, weights='distance')

clf.fit(vectors, y)

print(classification\_report(y\_test, clf.predict(vectors\_test)))

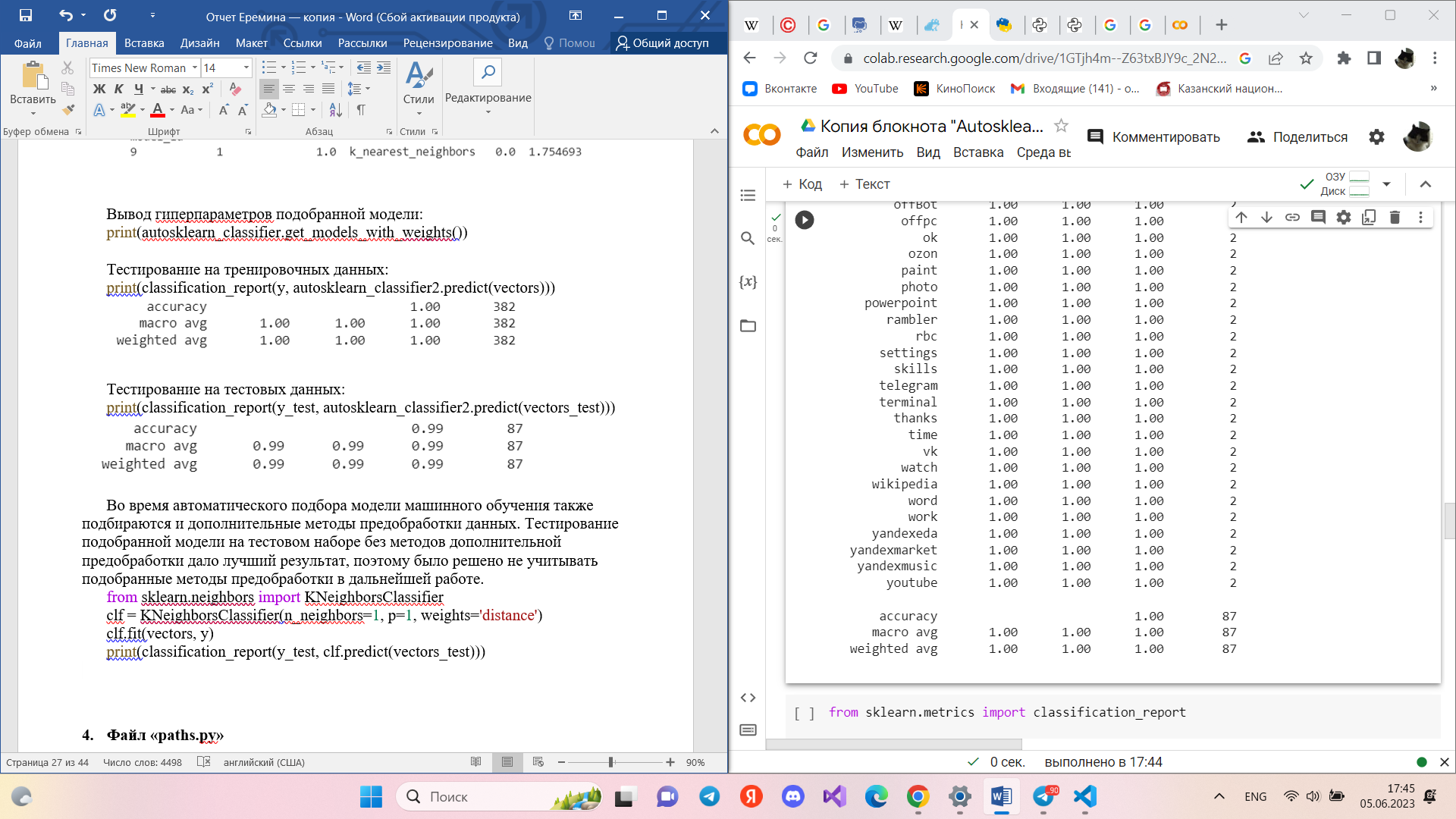


Рисунок 19 – Оценка модели на тестовых данных без учета предобработки.

* + 1. Модуль редактирования настроек

Параметры настроек хранятся в json-файлах, имеющих следующую структуру:

* «settings.json»: {'city': 'Казань'}
* «app\_paths.json»: {'calc': 'C:/Windows/System32/calc.exe', …}
* «web\_paths.json»: {'mail': 'https://mail.google.com/', …}

Взаимодействие с json-файлами осуществляется с помощью модуля «editing\_settings.py».

Импортирование модуля для работы с файлами:

import json

Загрузка данных из «settings.json»:

try:

with open ('json/settings.json', 'r') as filehandle:

SETTINGS = json.load(filehandle)

except:

SETTINGS={'city': ''}

Загрузка данных из «app\_paths.json»:

try:

with open ('json/app\_paths.json', 'r') as filehandle:

APP\_PATHS = json.load(filehandle)

except:

APP\_PATHS = {'calc': '', …}

Загрузка данных из «web\_paths.json»:

try:

with open ('json/web\_paths.json', 'r') as filehandle:

WEB\_PATHS = json.load(filehandle)

except:

WEB\_PATHS = {'mail': 'https://mail.google.com/', …}

В случае неудавшейся загрузки данных, значения переменных устанавливаются по умолчанию.

Функции сохранения настроек считывают введенные пользователем данные и сохраняют изменения в файлы json.

def save\_settings(self):

SETTINGS['city'] = self.city\_lineEdit.text()

with open('json/settings.json', 'w') as filehandle:

json.dump(SETTINGS, filehandle)

def save\_apps(self):

APP\_PATHS['calc'] = self.calc\_lineEdit.text();

…

with open('json/app\_paths.json', 'w') as filehandle:

json.dump(APP\_PATHS, filehandle)

def save\_webs(self):

WEB\_PATHS['mail'] = self.mail\_lineEdit.text()

…

with open('json/web\_paths.json', 'w') as filehandle:

json.dump(WEB\_PATHS, filehandle)

* + 1. Модуль синтеза речи

Синтез речи, озвучка, а также вывод ответа в текстовой форме реализованы в модуле «speech.py».

Импортирование модулей:

import pyttsx3 #модуль синтеза речи

from time import sleep #модуль для добавления задержки

from threading import \* #модуль для реализации многопоточности

Создание глобальной переменной-флага для блокировки микрофона и инициализация голосового движка:

mic\_blocked = False

engine = pyttsx3.init()

engine.setProperty('rate', 180)

Функции блокировки и разблокировки микрофона:

def block\_mic(): #блокировка микрофона

global mic\_blocked

mic\_blocked = True

def unblock\_mic(): #разблокировка микрофона

global mic\_blocked

mic\_blocked = False

Функция вывода символов на экран с задержкой:

def sleep\_print(str, label, text):

for char in text:

str+=char

label.setText(str)

label.adjustSize()

sleep(0.05)

Функция озвучки ответа и вывода текста на экран:

def speaker(self, text): #озвучка текста

t = Thread(target=sleep\_print, args=("", self.output\_label, text))

t.start()

block\_mic()

engine.say(text)

engine.runAndWait()

unblock\_mic()

* + 1. Модуль функций

Импортирование библиотек:

import sys #модуль, который обеспечивает доступ к некоторым переменным и функциям, взаимодействующим с интерпретатором python.

import requests #модуль для языка Python, используемый для упрощения работы с HTTP-запросами.

from translate import Translator #модуль для перевода текста

import webbrowser #модуль для просмотра веб-документов

import os #модуль для работы с операционной системой

import datetime #модуль для работы с датами и временем

Программные модули:

import editing\_settings

import speech

Функция отключения помощника с заменой кнопки остановки на кнопку запуска:

def offBot(self):

self.launch\_Button.setText("Пуск")

sys.exit()

Реализация функции запроса погоды:

def get\_weather(self):

Токен доступа для запроса:

open\_weather\_token = "a5635055be9774223fa602e7993d17de"

Инициализация переводчика:

translator = Translator(from\_lang="russian",to\_lang="english")

Перевод названия города на английский язык для формирования запроса:

try:

city = translator.translate(editing\_settings.SETTINGS['city'])

except:

speech.speaker(self, "Проверьте название города")

return

Словарь для вывода погоды на русском языке:

weather = { "Clear": "Ясно",

"Clouds": "Облачно",

"Rain": "Дождь",

"Drizzle": "Дождь",

"Thunderstorm": "Гроза",

"Snow": "Снег",

"Mist": "Туман"}

Запрос:

r = requests.get(f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={open\_weather\_token}&units=metric")

data = r.json()

Обработка результатов запроса:

city = data["name"]

cur\_weather = data["main"]["temp"]

weather\_description = data["weather"][0]["main"]

if weather\_description in weather:

wd = weather[weather\_description]

else:

wd = "не найдена"

humidity = data["main"]["humidity"]

pressure = data["main"]["pressure"]

wind = data["wind"]["speed"]

Формирование ответа:

result=(f"Погода в городе {editing\_settings.SETTINGS['city']} сегодня: {wd} \nТемпература {cur\_weather} градусов Цельсия\n"

f"Влажность {humidity}%\nДавление {pressure} миллиметров ртутного столба\nВетер: {wind} метров в секунду\n")

Вызов функции для озвучивания ответа:

speech.speaker(self, result)

Блоки запроса и формирования ответа заключены в конструкцию try-except:

try:

…

except:

mic.speaker("Проверьте название города")

Функция запроса времени:

def time(self):

now = datetime.datetime.now()

res = f" время: {now.hour}:{now.minute}"

speech.speaker(self, res)

Функция запуска программ:

def open\_app(self, app\_name):

ex=os.system(editing\_settings.APP\_PATHS[app\_name])

if ex != 0:

speech.speaker(self, "Исполняемый файл не найден")

Функция для открытия сайтов:

def open\_website(ws\_name):

webbrowser.open\_new\_tab(editing\_settings.WEB\_PATHS[ws\_name])

* + 1. Модуль обработки речи

Импортирование необходимых модулей:

* TF-IDF векторайзер из библиотеки Scikit-Learn

from sklearn.feature\_extraction.text import TfidfVectorizer

* LinearSVC – классификатор, реализующий метод k-ближайших соседей из библиотеки Sklearn.Neighbors

from sklearn.neighbors import KNeighborsClassifier

* Sounddevice – библиотека, позволяющая записывать аудио с помощью Python.

import sounddevice as sd

* Vosk – инструмент для распознавания речи. Также необходимо скачать модель vosk-model-small-ru-0.22 с официального сайта библиотеки vosk.

import vosk

* Random – модуль, предоставляющий функции для генерации случайных чисел, букв, случайного выбора элементов последовательности.

import random

* Json – модуль для кодирования и декодирования данных в удобном формате.

import json

* Queue – библиотека для реализации очередей с несколькими производителями и потребителями.

import queue

* Mystem из Pymystem3 – модуль, позволяющий лемматизировать текст на русском языке.

from pymystem3 import Mystem

* Threading – библиотека для реализации многопоточности

from threading import \*

* Программные модули:

import words

from functions import \*

import speech

Инициализация очереди:

q = queue.Queue()

Загрузка модели для распознавания речи:

model = vosk.Model('model\_small')

Выбор устройства для записи звука:

device = sd.default.device

Определение частоты микрофона:

samplerate = int(sd.query\_devices(device[0], 'input')['default\_samplerate'])

Инициализация лемматизатора:

myst = Mystem()

Функция лемматизации текста:

def diction\_form(text):

text = ''.join(myst.lemmatize(text)).rstrip('\n')

return text

Функция добавления в очередь символов из потока при записи речи:

def callback(indata, frames, time, status):

if not mic.mic\_blocked:

q.put(bytes(indata))

Основная функция обработки речи:

def main(self):

Создание листов для обучения модели и загрузка данных из json-файлов:

x = []

y = []

try:

with open('json/x.json', 'r') as filehandle:

x=json.load(filehandle)

with open('json/y.json', 'r') as filehandle:

y=json.load(filehandle)

except:

speech.speaker(self, "Не получается загрузить данные")

function.offBot(self)

В случае отсутствия файлов с данными, происходит озвучивание ошибки и завершение работы.

Изучение словарного запаса и векторизация данных:

vectorizer = TfidfVectorizer()

vectors = vectorizer.fit\_transform(x)

Обучение модели:

try:

clf = KNeighborsClassifier(n\_neighbors=1, p=1, weights='distance')

clf.fit(vectors, y)

except:

speech.speaker(self, "Загружены неверные данные")

offBot(self)

В случае загрузки некорректных данных, озвучивается соответствующая ошибка и завершается работа программы.

В цикле записи речь распознается в текст, лемматизируется и передается в функцию распознавания намерения. Также происходит вывод промежуточного результата распознавания для наглядности процесса и адаптация размера ярлыков под размер текста:

with sd.RawInputStream(samplerate=samplerate, blocksize=16000, device=device[0], dtype='int16', channels=1, callback=callback):

rec = vosk.KaldiRecognizer(model, samplerate)

temp1= "" #переменная для записи промежуточных результатов

while self.flag:

data = q.get()

temp2=rec.PartialResult() #считывание промежуточного результата

if temp2 != '{"text": ""}': #проверка существования промежуточного результата

temp1=check(rec, temp1, self) #обработка и вывод промежуточного результата

if rec.AcceptWaveform(data): #проверка на оконченную фразу

data = json.loads(rec.Result())['text'] #считывание результата

temp1="" #очистка переменной для записи промежуточных результатов

self.input\_label.setText(data) #вывод окончательно фразы

self.input\_label.adjustSize() #адаптация размера ярлыка

if data:

self.output\_label.setText("") #очистка ярлыка

self.output\_label.adjustSize() #адаптация размера ярлыка

recognize(self, diction\_form(data), vectorizer, clf) #вызов функции распознавания намерения

self.flag = True

offBot(self) #отключение

Функция обработки и вывода промежуточного результата распознавания речи:

def check(rec, temp1, self):

temp2=rec.PartialResult()

if temp2 != '{"text": ""}' and temp2 != '{\n  "partial" : ""\n}':

temp2 = temp2.replace('{\n  "partial" : "', "")

temp2 = temp2.replace('"\n}', "")

str= temp2[len(temp1):]

if temp2 != temp1:

speech.sleep\_print(temp1, self.input\_label, str)

return temp2

return ""

В функции распознавания намерения в первую очередь происходит поиск слова-триггера. В случае, если триггер найден, он удаляется, текст векторизуется, и предсказывается намерение пользователя. Затем выполняется функция, соответствующая классифицированному намерению и озвучивается ответ.

def recognize(self, data, vectorizer, clf): #анализ распознанной речи

trg = words.TRIGGERS.intersection(data.split()) #поиск слов-триггеров

if not trg:

return

data.replace(list(trg)[0], '') #удаление слов-триггеров

text\_vector = vectorizer.transform([data]).toarray()[0] #векторизация

intent = clf.predict([text\_vector])[0] #предсказание намерения

if intent not in words.SPEAKERS:

try:

out = random.choice(words.INTENTS[intent]["responses"]) #выбор рандомного ответа на предсказанное намерение

speech.speaker(self, out) #озвучка ответа

except:

speech.speaker(self, "Команда не распознана")

if intent in words.FUNCS: #выполнение функций

exec(intent + '(self)')

if intent in words.APPS:

open\_app(self, intent)

if intent in words.WEB\_SITES:

open\_website(intent)

Полный код программы приведен в приложении 1.

# Тестирование голосового помощника

На рисунке 20 представлен интерфейс программы при запуске.

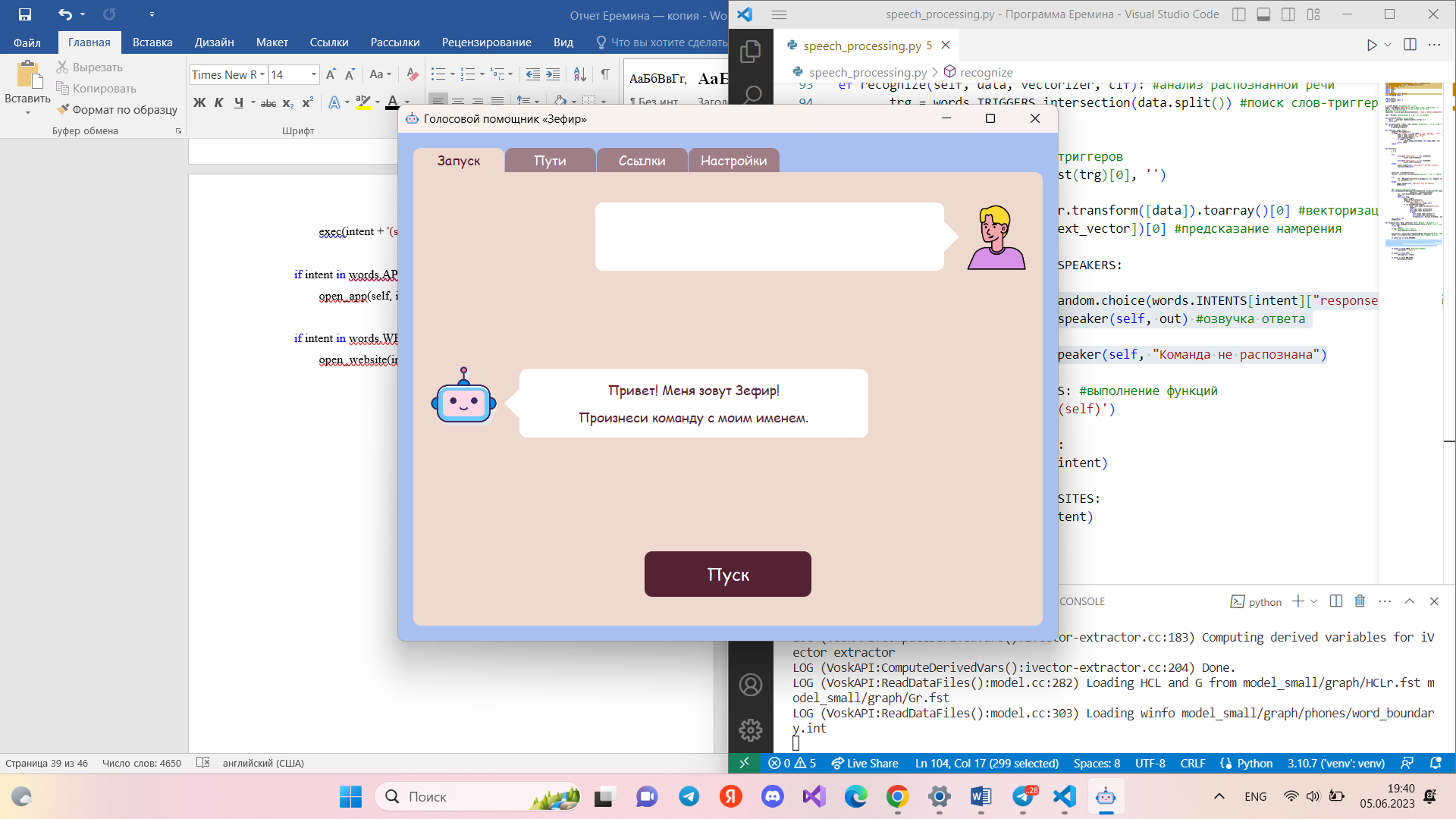


Рисунок 20 – Интерфейс программы.

При нажатии на кнопку «Пуск» и озвучивании команды со словом-триггером помощник выполняет команду, озвучивает ответ и выводит его в виде текста на экран. Распознанный текст также выводится на интерфейсе программы. Примеры работы программы представлены на рисунках 21-24.

Во вкладках «Пути», «Ссылки» и «Настройки» можно изменять параметры и сохранять путем нажатия кнопки «Сохранить». В качестве примера на рисунке 25 представлен вид вкладки «Пути». При перезапуске помощника, команды выполняются в соответствии с сохраненными изменениями.

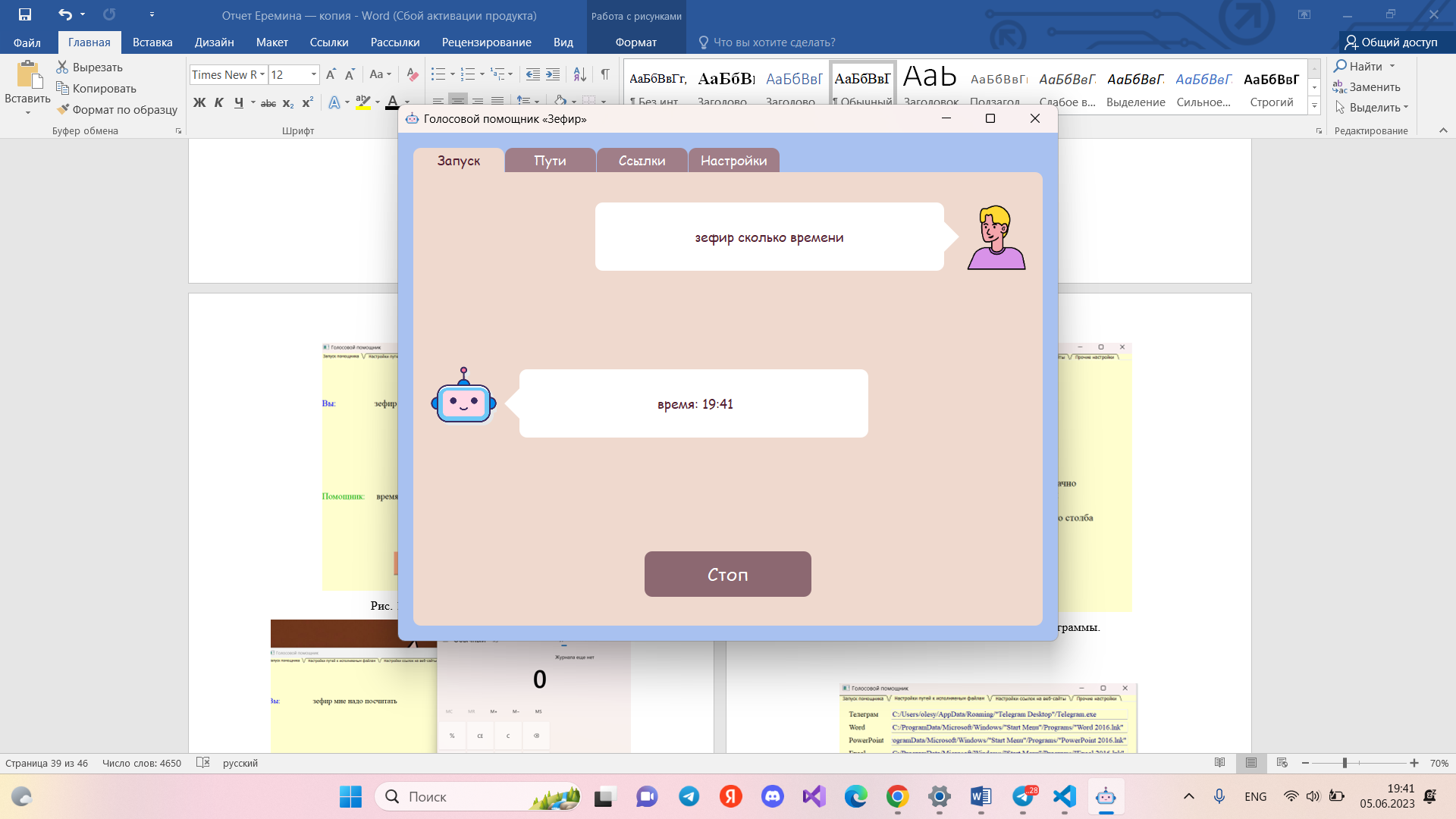


Рисунок 21 – Пример выполнения работы программы.

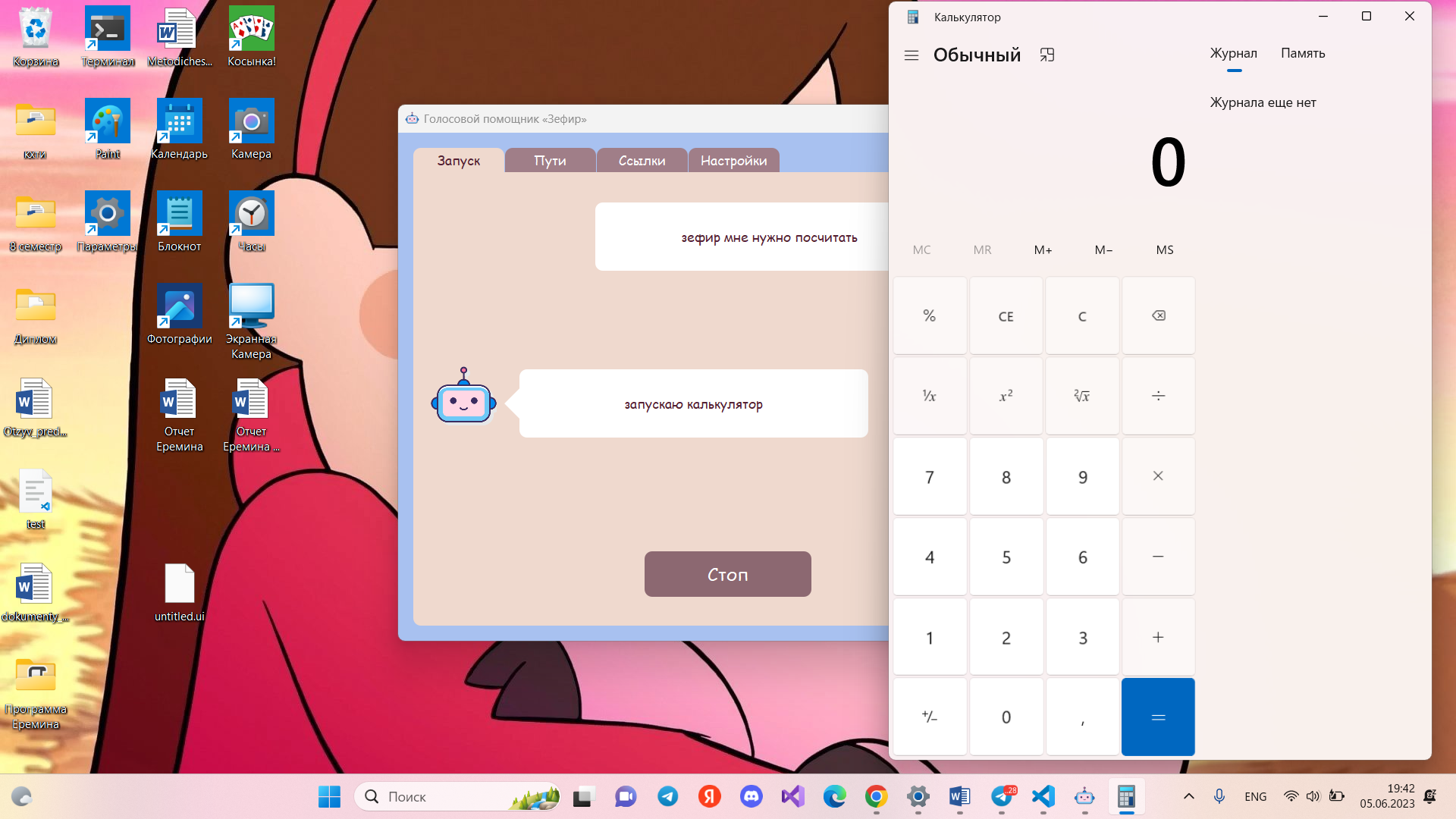


Рисунок 22 – Пример работы программы.

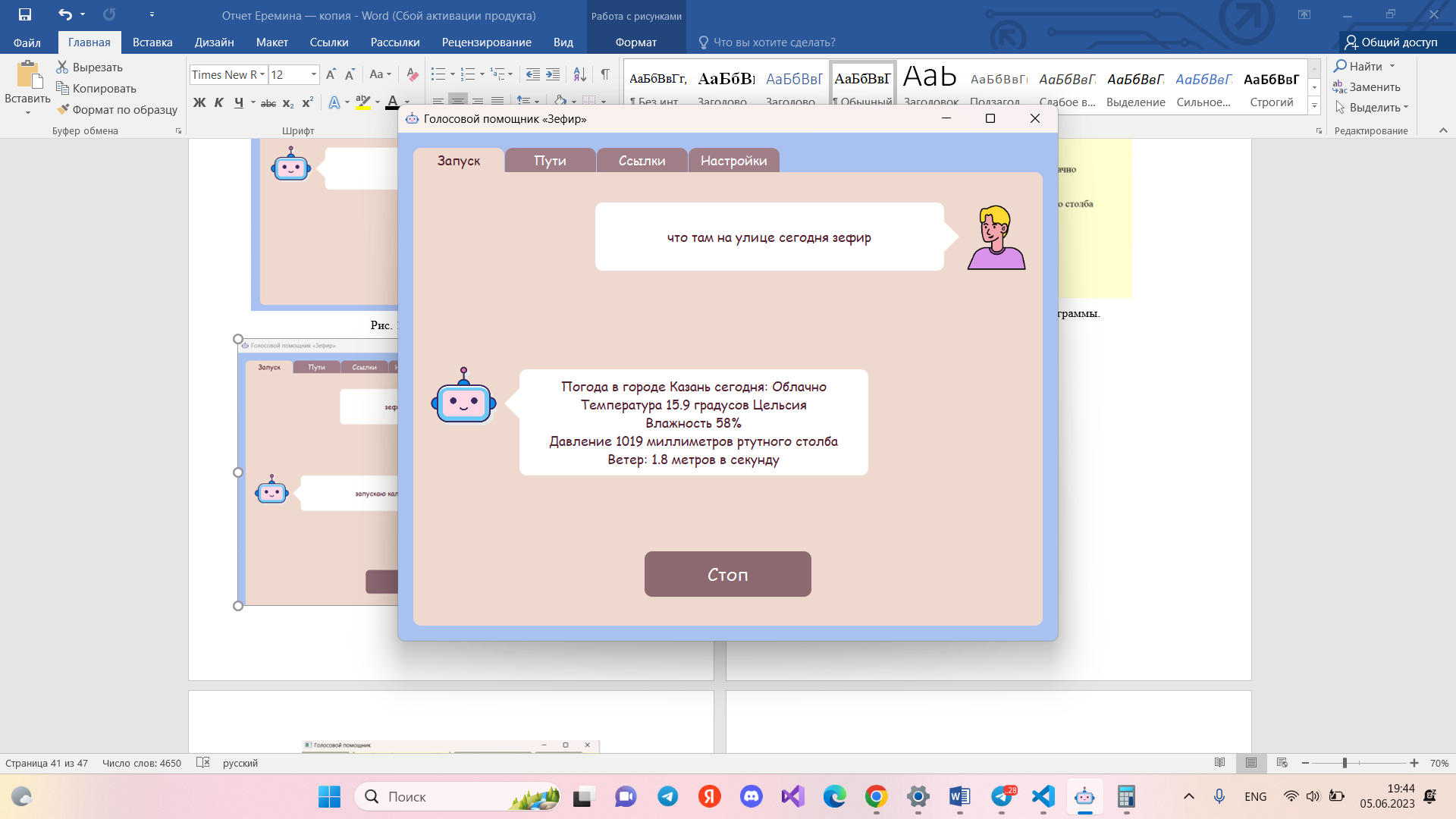


Рисунок 23 – Пример работы программы.

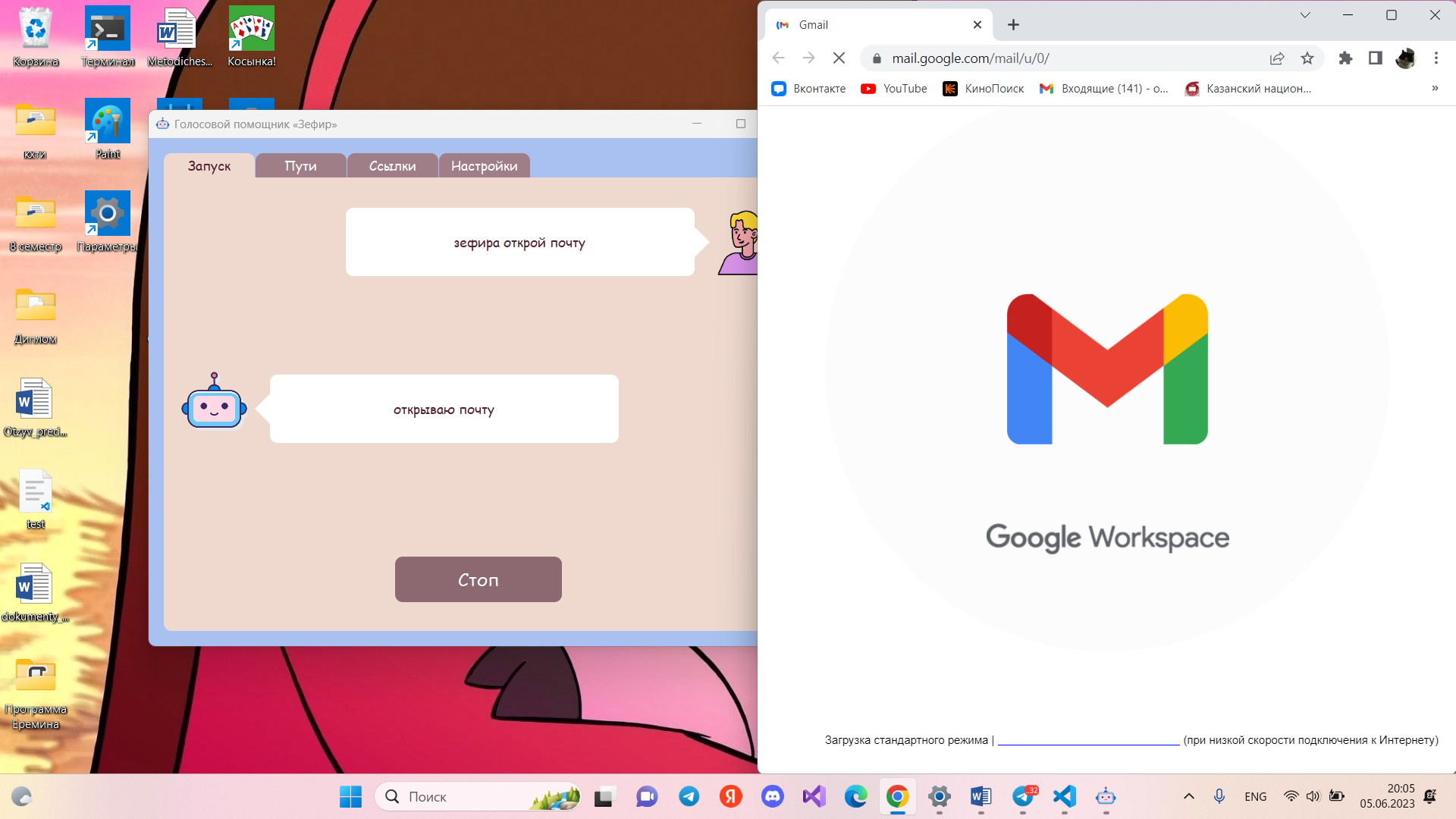


Рисунок 24 – Пример работы программы.

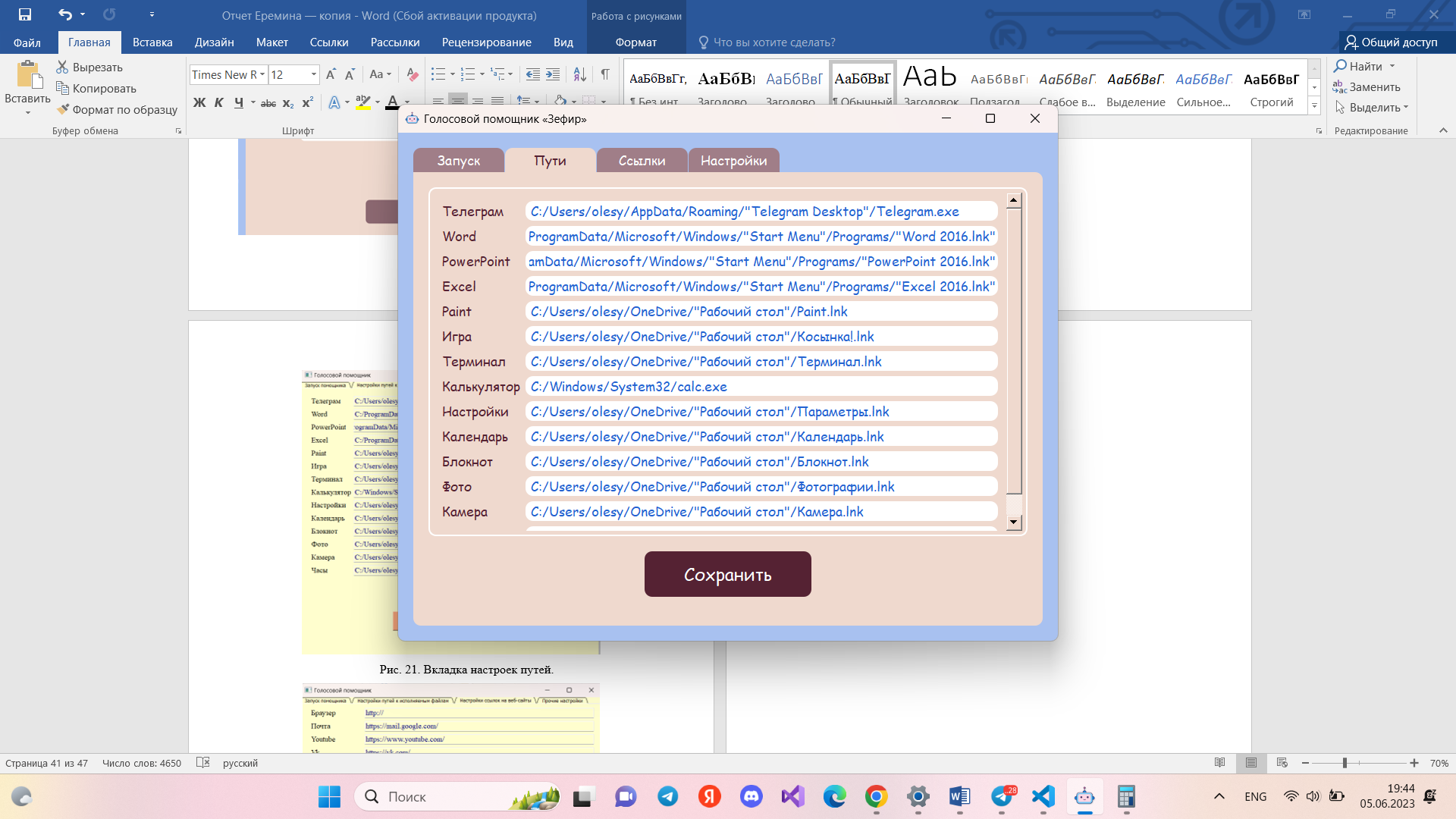


Рисунок 25 – Вкладка настроек путей.

# ЗАКЛЮЧЕНИЕ

В ходе работы над созданием голосового помощника для персонального компьютера были реализованы распознавание и синтез речи с помощью модулей Vosk и Pyttsx3, решена задача классификации намерений пользователя голосового помощника, реализована предобработка текста с помощью Mystem и TF-IDF векторизация, обучена модель на основе метода k-ближайших соседей с помощью Scikit-learn, реализован функционал голосового помощника и графический интерфейс с помощью фреймворка QT.

В процессе развития проекта планируется реализация следующих задач: расширение функционала голосового помощника, реализация возможности добавления пользователем команд, добавление возможности выбора сервисов для распознавания и синтеза речи, улучшение интерфейса. В случае значительного расширения функционала необходима реализация векторизации на основе эмбеддингов и классификации с помощью нейронной сети.

# СПИСОК ЛИТЕРАТУРЫ

1. Алиса (голосовой помощник) [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/А>лиса\_(голосовой\_помощник) (Дата обращения: 17.05.23)
2. Google Assistant [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Google_Assistant> (Дата обращения: 17.05.23)
3. Маруся (голосовой помощник) [Электронный ресурс] – URL: [https://ru.wikipedia.org/wiki/Маруся\_(голосовой\_помощник)](https://ru.wikipedia.org/wiki/Маруся_(голосовой_помощник)t) (Дата обращения: 17.05.23)
4. Amazon Alexa [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Amazon_Alexa> (Дата обращения: 17.05.23)
5. Олег (голосовой помощник) [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Олег_(голосовой_помощник)> (Дата обращения: 17.05.23)
6. Салют (голосовой помощник) [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Салют_(голосовой_помощник)> (Дата обращения: 17.05.23)
7. Кортана (голосовой помощник) [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Кортана_(голосовой_помощник)> (Дата обращения: 17.05.23)
8. Siri [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Siri> (Дата обращения: 17.05.23)
9. Laitis [Электронный ресурс] – URL: <https://laitis.ru> (Дата обращения: 17.05.23)
10. Задача классификации [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Задача_классификации> (Дата обращения: 17.05.23)
11. Распознавание речи [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Распознавание_речи> (Дата обращения: 17.05.23)
12. Синтез речи [Электронный ресурс] – URL: <https://neerc.ifmo.ru/wiki/index.php?title=Синтез_речи> (Дата обращения: 17.05.23)
13. Акжолов Р.К. Предобработка текста для решения задач NLP / Р.К. Акжолов, А.В. Верига // Вестник науки. – 2020. - №3. – С.66-68
14. 4 метода векторизации текстов. [Электронный ресурс] – URL: <https://medium.com/@bigdataschool/4-метода-векторизации-текстов-f8ac90e4175a> (Дата обращения: 17.05.23)
15. Чудесный мир Word Embeddings: какие они бывают и зачем нужны? [Электронный ресурс] – URL: <https://habr.com/ru/company/ods/blog/329410/>(Дата обращения: 17.05.23)
16. Обзор методов классификации в машинном обучении с помощью Scikit-Learn. [Электронный ресурс] – URL: <https://tproger.ru/translations/scikit-learn-in-python/> (Дата обращения: 17.05.23)
17. Автоматическое машинное обучение [Электронный ресурс] – URL: <https://neerc.ifmo.ru/wiki/index.php?title=Автоматическое_машинное_обучение> (Дата обращения: 17.05.23)
18. NLP – Преобразование текста: Word2Vec. [Электронный ресурс] – URL: <https://habr.com/ru/company/otus/blog/574624/> (Дата обращения: 17.05.23)
19. Deep Learning vs common sense: разрабатываем чат-бота. [Электронный ресурс] – URL: <https://habr.com/ru/company/oleg-bunin/blog/455652/> (Дата обращения: 17.05.23)
20. Дерево решений. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Дерево_решений> (Дата обращения: 17.05.23)
21. Линейный дискриминантный анализ. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Линейный_дискриминантный_анализ> (Дата обращения: 17.05.23)
22. Логистическая регрессия. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Логистическая_регрессия> (Дата обращения: 17.05.23)
23. Метод опорных векторов. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Метод_опорных_векторов> (Дата обращения: 17.05.23)
24. Метод k-ближайших соседей. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Метод_k-ближайших_соседей> (Дата обращения: 17.05.23)
25. Наивный байесовский классификатор. [Электронный ресурс] – URL: <https://ru.wikipedia.org/wiki/Наивный_байесовский_классификатор> (Дата обращения: 17.05.23)
26. Обучение с учителем. [Электронный ресурс] – URL: <https://scikit-learn.ru/category/supervised_learning/> (Дата обращения: 17.05.23)
27. Вьюгин В.В. Математические основы машинного обучения и прогнозирования / В.В. Вьюгин. – М.: МЦНМО, 2018. - 384 с.
28. Воронина В.В. Теория и практика машинного обучения / В.В. Воронина, А.В. Михеев, Н.Г. Ярушкина, К.В. Святов – Ульяновск: УлГТУ, 2017. – 290с.

Приложение 1

Код программы

«editing\_settings.py»

import json

try:

    with open ('json/app\_paths.json', 'r') as filehandle:

        APP\_PATHS = json.load(filehandle)

except:

    APP\_PATHS = {

    'telegram': '',

    'calc': '',

    'game': '',

    'excel': '',

    'paint': '',

    'powerpoint': '',

    'terminal': '',

    'word': '',

    'settings': '',

    'calendar': '',

    'notebook': '',

    'photo': '',

    'camera': '',

    'watch': ''

}

try:

    with open ('json/web\_paths.json', 'r') as filehandle:

        WEB\_PATHS = json.load(filehandle)

except:

    WEB\_PATHS = {

    'browser': 'http://',

    'mail': 'https://mail.google.com/',

    'youtube': 'https://www.youtube.com/',

    'vk': 'https://vk.com/',

    'kinopoisk': 'https://www.kinopoisk.ru/',

    'yandexmusic': 'https://music.yandex.ru/',

    'wikipedia': 'https://ru.wikipedia.org/',

    'ok': 'https://ok.ru/',

    'google': 'https://www.google.ru/',

    'rambler': 'https://www.rambler.ru/',

    'avito': 'https://www.avito.ru/',

    'gismeteo': 'https://www.gismeteo.ru/',

    'ozon': 'https://www.ozon.ru/',

    'rbc': 'https://rt.rbc.ru/',

    'yandexmarket': 'https://market.yandex.ru/',

    'gosuslugi': 'https://www.gosuslugi.ru/',

    'yandexeda': 'https://eda.yandex.ru/',

    'googletranslate': 'https://translate.google.com/'

}

try:

    with open ('json/settings.json', 'r') as filehandle:

        SETTINGS = json.load(filehandle)

except:

    SETTINGS={'city': ''}

def save\_apps(self):

        APP\_PATHS['telegram'] = self.telegram\_lineEdit.text();

        APP\_PATHS['calc'] = self.calc\_lineEdit.text();

        APP\_PATHS['game'] = self.game\_lineEdit.text();

        APP\_PATHS['excel'] = self.excel\_lineEdit.text();

        APP\_PATHS['paint'] = self.paint\_lineEdit.text();

        APP\_PATHS['powerpoint'] = self.powerpoint\_lineEdit.text();

        APP\_PATHS['terminal'] = self.terminal\_lineEdit.text();

        APP\_PATHS['word'] = self.word\_lineEdit.text();

        APP\_PATHS['settings'] = self.settings\_lineEdit.text();

        APP\_PATHS['calendar'] = self.calendar\_lineEdit.text();

        APP\_PATHS['notebook'] = self.notebook\_lineEdit.text();

        APP\_PATHS['photo'] = self.photo\_lineEdit.text();

        APP\_PATHS['camera'] = self.camera\_lineEdit.text();

        APP\_PATHS['watch'] = self.watch\_lineEdit.text();

        with open('json/app\_paths.json', 'w') as filehandle:

                json.dump(APP\_PATHS, filehandle)

def save\_webs(self):

        WEB\_PATHS['browser'] = self.browser\_lineEdit.text()

        WEB\_PATHS['mail'] = self.mail\_lineEdit.text()

        WEB\_PATHS['youtube'] = self.youtube\_lineEdit.text()

        WEB\_PATHS['vk'] = self.vk\_lineEdit.text()

        WEB\_PATHS['kinopoisk'] = self.kinopoisk\_lineEdit.text()

        WEB\_PATHS['yandexmusic'] = self.yandexmusic\_lineEdit.text()

        WEB\_PATHS['wikipedia'] = self.wikipedia\_lineEdit.text()

        WEB\_PATHS['ok'] = self.ok\_lineEdit.text()

        WEB\_PATHS['google'] = self.google\_lineEdit.text()

        WEB\_PATHS['rambler'] = self.rambler\_lineEdit.text()

        WEB\_PATHS['avito'] = self.avito\_lineEdit.text()

        WEB\_PATHS['gismeteo'] = self.gismeteo\_lineEdit.text()

        WEB\_PATHS['ozon'] = self.ozon\_lineEdit.text()

        WEB\_PATHS['rbc'] = self.rbc\_lineEdit.text()

        WEB\_PATHS['yandexmarket'] = self.yandexmarket\_lineEdit.text()

        WEB\_PATHS['gosuslugi'] = self.gosuslugi\_lineEdit.text()

        WEB\_PATHS['yandexeda'] = self.yandexeda\_lineEdit.text()

        WEB\_PATHS['googletranslate'] = self.googletranslate\_lineEdit.text()

        with open('json/web\_paths.json', 'w') as filehandle:

                json.dump(WEB\_PATHS, filehandle)

def save\_settings(self):

        SETTINGS['city'] = self.city\_lineEdit.text()

        with open('json/settings.json', 'w') as filehandle:

                json.dump(SETTINGS, filehandle)

«functions.py»

import sys #модуль, который обеспечивает доступ

            #к некоторым переменным и функциям,

            # #взаимодействующим с интерпретатором python.

import requests #модуль для языка Python, который используют

                #для упрощения работы с HTTP-запросами.

from translate import Translator #модуль для перевода

import webbrowser #модуль для просмотра веб-документов

import os #модуль для работы с операционной системой

import datetime #модуль для работы с датами и временем

import editing\_settings

import speech

def offBot(self): #функция отключения бота

    self.launch\_Button.setText("Пуск")

    self.launch\_Button.setStyleSheet("QPushButton\n"

                                    "{\n"

                                    "    color: rgb(255, 255, 255);\n"

                                    "    background-color: rgb(85, 34, 51);\n"

                                    "    font: 14pt \"Comic Sans MS\";\n"

                                    "    border-radius: 10;\n"

                                    "}\n"

                                    "\n"

                                    "QPushButton:hover\n"

                                    "{\n"

                                    "    background-color: rgb(177, 149, 158);\n"

                                    "}")

    sys.exit()

def get\_weather(self):

    open\_weather\_token = "a5635055be9774223fa602e7993d17de"

    translator = Translator(from\_lang="russian",to\_lang="english")

    try:

        city = translator.translate(editing\_settings.SETTINGS['city'])

    except:

        speech.speaker(self, "Проверьте название города")

        return

    weather = {

        "Clear": "Ясно",

        "Clouds": "Облачно",

        "Rain": "Дождь",

        "Drizzle": "Дождь",

        "Thunderstorm": "Гроза",

        "Snow": "Снег",

        "Mist": "Туман"

    }

    try:

        r = requests.get(

            f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={open\_weather\_token}&units=metric"

        )

        data = r.json()

        city = data["name"]

        cur\_weather = data["main"]["temp"]

        weather\_description = data["weather"][0]["main"]

        if weather\_description in weather:

            wd = weather[weather\_description]

        else:

            wd = "не найдена"

        humidity = data["main"]["humidity"]

        pressure = data["main"]["pressure"]

        wind = data["wind"]["speed"]

        result=(f"Погода в городе {editing\_settings.SETTINGS['city']} сегодня: {wd} \nТемпература {cur\_weather} градусов Цельсия\n"

              f"Влажность {humidity}%\nДавление {pressure} миллиметров ртутного столба\nВетер: {wind} метров в секунду"

              )

        speech.speaker(self, result)

    except Exception as ex:

        speech.speaker(self,"Проверьте название города")

def time(self):

    now = datetime.datetime.now()

    res = f" время: {now.hour}:{now.minute}"

    speech.speaker(self, res)

def open\_app(self, app\_name):

    ex=os.system(editing\_settings.APP\_PATHS[app\_name])

    if ex != 0:

        speech.speaker(self, "Исполняемый файл не найден")

def open\_website(ws\_name):

    webbrowser.open\_new\_tab(editing\_settings.WEB\_PATHS[ws\_name])

def offpc():

    os.system('shutdown -s')

«lemmatization.py»

import json

from pymystem3 import Mystem

import words

myst = Mystem()

def diction\_form(text): #лемматизация

    text = ''.join(myst.lemmatize(text)).rstrip('\n')

    return text

x = []

y = []

for intent in words.INTENTS: #заполняем листы примерами входных текстов и соответствующими намерениями

    print("Гружусь...")

    examples = words.INTENTS[intent]["examples"]

    for example in examples:

        word= diction\_form(example)

        x.append(word)

        y.append(intent)

with open('json/x.json', 'w') as filehandle:

    json.dump(x, filehandle)

with open('json/y.json', 'w') as filehandle:

    json.dump(y, filehandle)

«speech\_processing.py»

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.neighbors import KNeighborsClassifier

import sounddevice as sd

import vosk

import random

import json

import queue

from pymystem3 import Mystem

from threading import \*

import words

from functions import \*

import speech

q = queue.Queue() #инициализируем очередь

model = vosk.Model('model\_small') #подгружаем модель распознования речи

device = sd.default.device # выбираем девайсы по умолчанию

#получаем частоту микрофона

samplerate = int(sd.query\_devices(device[0], 'input')['default\_samplerate'])

myst = Mystem() #инициализация программы для лемматизации

def diction\_form(text): #лемматизация

    text = ''.join(myst.lemmatize(text)).rstrip('\n')

    return text

def callback(indata, frames, time, status): #добавление в очередь семплов из потока

    if not speech.mic\_blocked:

        q.put(bytes(indata))

def check(rec, temp1, self):

        temp2=rec.PartialResult()

        if temp2 != '{"text": ""}' and temp2 != '{\n  "partial" : ""\n}':

                temp2 = temp2.replace('{\n  "partial" : "', "")

                temp2 = temp2.replace('"\n}', "")

                str= temp2[len(temp1):]

                if temp2 != temp1:

                        speech.sleep\_print(temp1, self.input\_label, str)

                return temp2

        return ""

def main(self):

        x = []

        y = []

        try:

                with open('json/x.json', 'r') as filehandle:

                        x=json.load(filehandle)

                with open('json/y.json', 'r') as filehandle:

                        y=json.load(filehandle)

        except:

                speech.speaker(self, "Не получается загрузить данные")

                function.offBot(self);

        vectorizer = TfidfVectorizer()

        vectors = vectorizer.fit\_transform(x) #изучение словарного запаса и idf; преобразование документов в матрицу

        try:

                clf = KNeighborsClassifier(n\_neighbors=1, p=1, weights='distance')

                clf.fit(vectors, y)

        except:

               speech.speaker(self, "Загружены неверные данные")

               offBot(self)

        #постоянная прослушка микрофона

        with sd.RawInputStream(samplerate=samplerate, blocksize=16000, device=device[0], dtype='int16',

                                channels=1, callback=callback):

                rec = vosk.KaldiRecognizer(model, samplerate)

                temp1=""

                while self.flag:

                        data = q.get()

                        temp2=rec.PartialResult()

                        if temp2 != '{"text": ""}':

                                temp1=check(rec, temp1, self)

                        if rec.AcceptWaveform(data):

                                data = json.loads(rec.Result())['text']

                                temp1=""

                                self.input\_label.setText(data)

                                self.input\_label.adjustSize()

                                if data:

                                    self.output\_label.setText("")

                                    self.output\_label.adjustSize()

                                    recognize(self, diction\_form(data), vectorizer, clf) #обработка

        self.flag = True

        offBot(self)

def recognize(self, data, vectorizer, clf): #анализ распознанной речи

        trg = words.TRIGGERS.intersection(data.split()) #поиск слов-триггеров

        if not trg:

                return

        if trg: #удаление слов-триггеров

                data.replace(list(trg)[0], '')

        text\_vector = vectorizer.transform([data]).toarray()[0] #векторизация

        intent = clf.predict([text\_vector])[0] #предсказание намерения

        if intent not in words.SPEAKERS:

                try:

                        out = random.choice(words.INTENTS[intent]["responses"]) #выбор рандомного ответа на предсказанное намерение

                        speech.speaker(self, out) #озвучка ответа

                except:

                       speech.speaker(self, "Команда не распознана")

        if intent in words.FUNCS: #выполнение функций

                exec(intent + '(self)')

        if intent in words.APPS:

                open\_app(self, intent)

        if intent in words.WEB\_SITES:

                open\_website(intent)

«speech.py»

import pyttsx3

from time import sleep

from threading import \*

mic\_blocked = False #глобальная переменная-флаг для блокировки микрофона

engine = pyttsx3.init() #инициализируем голосовой движок

engine.setProperty('rate', 180)

def block\_mic(): #блокировка микрофона

    global mic\_blocked

    mic\_blocked = True

def unblock\_mic(): #разблокировка микрофона

    global mic\_blocked

    mic\_blocked = False

def sleep\_print(str, label, text):

    for char in text:

        str+=char

        label.setText(str)

        label.adjustSize()

        sleep(0.05)

def speaker(self, text): #озвучка текста

    t = Thread(target=sleep\_print, args=("", self.output\_label, text))

    t.start()

    block\_mic()

    engine.say(text)

    engine.runAndWait()

    unblock\_mic()

«words.py»

TRIGGERS = {'зефир', 'сфер', 'эфир', 'зефира', 'зверь', 'сефер'}

FUNCS={'offBot','get\_weather', 'time', 'offpc'}

APPS={'telegram', 'calc', 'game', 'excel', 'paint', 'powerpoint', 'terminal', 'word', 'settings', 'calendar', 'notebook', 'photo', 'camera', 'watch'}

WEB\_SITES={'browser', 'mail', 'youtube', 'vk', 'kinopoisk', 'yandexmusic', 'wikipedia', 'ok', 'google', 'rambler', 'avito', 'gismeteo', 'ozon', 'rbc', 'yandexmarket', 'gosuslugi', 'yandexeda', 'googletranslate'}

SPEAKERS={'get\_weather', 'time'}

INTENTS = {

    "hello": {

        "examples": ["привет", "приветик", "приветики", "здравствуйте", "здравствуй", "приветствую", "доброе утро", "добрый день", "добрый вечер"],

        "responses": ["привет", "здравствуйте"],

    },

    "offBot": {

        "examples": ["пока", "отключись", "до свидания", "до встречи", "прощай", "прощайте", "выключись", "отключайся", "выключайся", "заверши работу"],

        "responses": ["пока", "отключаюсь", "до свидания", "до встречи"],

    },

    "how\_are\_you": {

        "examples": ["как у тебя дела", "как твои дела" "как ты", "как твоё настроение", "как жизнь", "как день"],

        "responses": ["всё хорошо спасибо", "лучше всех", "нормально", "отлично"],

    },

    "thanks": {

        "examples": ["спасибо", "благодарю", "я тебе благодарна", "я тебе благодарен", "большое спасибо"],

        "responses": ["пожалуйста", "обращайтесь", "обращайся", "всегда рада помочь", "рада помочь"],

    },

    "get\_weather": {

        "examples": ["погода","какая погода", "сколько градусов", "какая погода на улице", "что с погодой", "сколько градусов на улице", "что там на улице", "какая сегодня погода", "подскажи погоду", "подскажи сколько градусов", "подскажи какая сегодня погода"],

        "responses": ["сейчас посмотрю", "сейчас гляну", "сейчас проверю"],

    },

    "browser": {

        "examples": ["открой браузер", "запусти браузер", "интернет", "открой новую вкладку в браузере", "открой интернет", "запусти интернет", "открой новую вкладку" ],

        "responses": ["открываю браузер"],

    },

    "game": {

        "examples": ["играть", "хочу играть", "включи игру", "игра", "хочу поиграть", "запусти игру"],

        "responses": ["включаю игру"],

    },

    "offpc": {

        "examples": ["выключи компьютер", "отключи компьютер", "отруби комп", "выключи комп", "отключи комп"],

        "responses": ["отключаю компьютер"],

    },

    "here": {

        "examples": ["ты тут", "ты здесь", "здесь", "ты где"],

        "responses": ["я тут", "я здесь"],

    },

    "work": {

        "examples": ["ты работаешь"],

        "responses": ["ещё как"],

    },

    "joke": {

        "examples": ["расскажи анектод", "анекдот", "пошути", "расскажи шутку", "рассмеши меня"],

        "responses": [f"А правда, что кошки и дрессировка несовместимы?\nНет, мой кот выдрессировал меня за три дня!", f"Народная мудрость гласит:\nодежду надо покупать под цвет шерсти своего кота.", f"Вчера решил посадить кошку на диету,\nа то она слишком толстая стала.\nСегодня она родила.\nА на что ты готов ради еды?"],

    },

    "telegram": {

        "examples": ["открой телеграмм", "запусти телеграмм", "открой телегу", "запусти телегу"],

        "responses": ["открываю телеграмм"],

    },

    "calc": {

        "examples": ["открой калькулятор", "запусти калькулятор", "надо посчитать"],

        "responses": ["запускаю калькулятор"],

    },

    "time": {

        "examples": ["время", "сколько времени", "сколько часов"],

        "responses": [""],

    },

    "mail": {

        "examples": ["открой почту", "хочу проверить почту", "письма", "хочу почитать письма", "хочу почитать почту"],

        "responses": ["открываю почту"],

    },

    "skills": {

        "examples": ["что ты умеешь", "что ты можешь", "твой функционал", "расскажи что ты умеешь", "расскажи что ты можешь"],

        "responses": ["Я умею узнавать погоду и время,\nоткрывать программы и сайты,\nотключать компьютер и рассказывать анектоды"],

    },

    "excel": {

        "examples": ["открой эксель", "запусти эксель", "открой иксель",  "открой иксэль", "запусти иксэль"],

        "responses": ["запускаю эксэль"],

    },

    "paint": {

        "examples": ["открой пэйнт", "запусти пэйнт"],

        "responses": ["запускаю пэйнт"],

    },

    "powerpoint": {

        "examples": ["открой повар поинт", "лавр поинт", "запусти повар понт", "открой павр поинт", "нужно сделать презентацию", "открой презентацию"],

        "responses": ["запускаю павер поинт"],

    },

    "terminal": {

        "examples": ["открой терминал", "запусти терминал", "открой командную строку"],

        "responses": ["запускаю терминал"],

    },

    "word": {

        "examples": ["открой ворд", "запусти борт", "открой ворт", "открой горд", "запусти ворлд", "открой ворота"],

        "responses": ["запускаю ворд"],

    },

    "settings": {

        "examples": ["открой настройки", "запусти параметры", "запусти настройки", "открой параметры"],

        "responses": ["открываю параметры"],

    },

    "calendar": {

        "examples": ["открой календарь", "запусти календарь"],

        "responses": ["открываю календарь"],

    },

    "notebook": {

        "examples": ["открой блокнот", "запусти блокнот"],

        "responses": ["открываю блокнот"],

    },

    "photo": {

        "examples": ["открой галерею", "запусти фотографии", "хочу посмотреть фотографии", "хочу посмотреть фотки", "открой фотографии"],

        "responses": ["открываю фотографии"],

    },

    "camera": {

        "examples": ["открой камеру", "запусти камеру","как я выгляжу"],

        "responses": ["открываю камеру"],

    },

    "watch": {

        "examples": ["открой часы", "запусти часы"],

        "responses": ["открываю часы"],

    },

    "youtube": {

        "examples": ["открой ютуб", "запусти ютюб", "зайди в ютуб"],

        "responses": ["открываю ютуб"],

    },

    "vk": {

        "examples": ["открой быка", "запусти века", "открой контакт", "запусти вконтакте", "зайди в контакт", "зайди в вака"],

        "responses": ["открываю вконтакте"],

    },

    "kinopoisk": {

        "examples": ["открой кинопоиск", "запусти кинопоиск", "хочу посмотреть кино", "глянуть фильм", "зайди на кинопоиск"],

        "responses": ["открываю кинопоиск"],

    },

    "yandexmusic": {

        "examples": ["открой яндекс музыку", "запусти музыку на яндексе", "зайди на яндекс музыку", "музыка", "включи музыку", "хочу послушать музыку", "хочу потанцевать"],

        "responses": ["открываю яндекс музыку"],

    },

    "wikipedia": {

        "examples": ["открой википидию", "запусти векипедию", "википедия", "зайди на википедию"],

        "responses": ["открываю википедию"],

    },

    "ok": {

        "examples": ["открой одноклассники", "запусти одноклассники", "зайди на одноклассники"],

        "responses": ["открываю одноклассники"],

    },

    "google": {

        "examples": ["открой гугл", "запусти гугл", "зайди в губы"],

        "responses": ["открываю гугл"],

    },

    "rambler": {

        "examples": ["открой рамблер", "запусти рамблер", "зайди в рамблер"],

        "responses": ["открываю рамблер"],

    },

    "avito": {

        "examples": ["открой авито", "запусти авито", "зайди в вито", "зайди в элита"],

        "responses": ["открываю авито"],

    },

    "gismeteo": {

        "examples": ["открой гисметео", "запусти гисметео", "зайди в гисметео", "покажи погоду на гисметео"],

        "responses": ["открываю гисметео"],

    },

    "ozon": {

        "examples": ["открой озон", "запусти озон", "зайди в озон", "надо сделать заказ на зоне", "заказ позвони"],

        "responses": ["открываю озон"],

    },

    "rbc": {

        "examples": ["открой рбк", "зайди в рбк"],

        "responses": ["открываю эр бэ ка"],

    },

    "yandexmarket": {

        "examples": ["открой яндекс маркет", "зайди в яндекс маркет "],

        "responses": ["открываю яндекс маркет"],

    },

    "gosuslugi": {

        "examples": ["открой госуслуги", "зайди в госуслуги "],

        "responses": ["открываю госуслуги"],

    },

    "yandexeda": {

        "examples": ["открой яндекс еду", "зайди в яндкс еду", "хочу кушать", "хочу есть"],

        "responses": ["открываю яндекс еду"],

    },

    "googletranslate": {

        "examples": ["открой гугл переводчик", "зайди в губы переводчик", "надо перевести", "нужно перевести", "переведи", "перевод"],

        "responses": ["открываю гугл переводчик"],

    },

}