Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Интеллектуальная система автоматизации работы онлайн-консультанта**

Студент гр. 43501/3 К.А. Галушко

Санкт-Петербург – 2017

Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологий
Кафедра компьютерных систем и программных технологий

Работа допущена к защите
зав. кафедрой

_____ В.М. Ицыксон

«____» _____ 2017 г.

ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Интеллектуальная система автоматизации работы онлайн-консультанта**

Направление: 09.03.01 – Информатика и вычислительная техника

Выполнил студент гр. 43501/3 _____ К.А. Галушко

Научный руководитель,
к. т. н., доц. _____ И.В. Стручков

Санкт-Петербург – 2017

РЕФЕРАТ

Дипломная работа, 57 стр., 19 рис., 5 ист., 4 прил.

Интеллектуальная система автоматизации работы онлайн-консультанта

В выпускной работе проводится разработка веб-приложения, которое является автоматизированным рабочим местом онлайн-консультанта. В работе проведён анализ существующих решений для автоматизации взаимодействий с клиентами. Описаны использованные технологии, процесс построения веб-приложения, модуль машинного обучения, взаимодействие с базой данных и системой обмена мгновенными сообщениями.

ABSTRACT

Graduate work, 57 pages, 19 figures, 5 references, 4 appendicies

INTELLIGENT COMPUTER-AIDED SYSTEM FOR ONLINE CONSULTANT

The document develops a computer-aided workplace for an online consultant. The paper analyzes the market of existing solutions for computer-aided for interactions with clients. Describes the technologies used, the process of building a web application, the machine learning module, the interaction with database and messenger.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Ежедневно растет число пользователей онлайн ресурсов относительно числа посетителей аналогичных сфер оффлайн. Речь идет о магазинах и их интернет версиях, о музыкальных концертах и их онлайн трансляциях, о местах питания и приложениях для доставки еды, а также о многих других сфера жизни. И важным этапом развития сетевых ресурсов является появление не только технической поддержки, но и также двустороннего канала связи между пользователями и представителями организации, а именно онлайн-консультации. Данный вид общения с клиентами стал не только средством помощи, но и инструментом для взаимодействия.

Основные причины внедрения автоматизированного рабочего места специалиста:

- Повышение качества работы консультантов;
- Значительное повышение производительности труда консультантов;
- Повышение лояльности пользователей;
- Увеличение прибыли ресурса;

В данный момент, существует множество автоматизированных систем поддержки принятия решений, а также программного обеспечения для онлайн консультации, однако, нет инструментов, которые объединяли бы два этих аспекта работы.

1. Анализ существующих систем автоматизации работы консультантов

1.1. Общие принципы работы онлайн-консультантов

В настоящее время все чаще вместо звонков по телефону, встреч и прочих традиционных способов клиенты предпочитают онлайн-консультации. Появилось множество каналов для этих целей: модули для сайтов, сервисы обмена мгновенными сообщениями с возможностью интеграции, CRM-системы и другие. Все они разбиваются на два подмножества, а именно, те, которые работают на стороне клиентов(фронт-офис), и те, которые работают на стороне консультантов(бэк-офис).

Под первым множеством можно принимать чаты в интерфейсе сайта и различные сервисы для обмена мгновенными сообщениями. Такая связь позволяет получить консультацию именно у специалистов ресурса. Каждый из видов средств для обмена информацией имеет свои как положительные, так и отрицательные стороны. Например, в случае с рядовыми и часто задаваемыми вопросами проще незамедлительно спросить в чате и получить быстрый ответ. Однако, чаты в интерфейса сайта часто некорректно отображаются с мобильных устройств или и вовсе являются непроизводительными. В данной работе активно используется канал связи с клиентом в виде сервиса обмена мгновенными сообщениями Telegram. Этот канал был выбран поскольку сервисы для обмена мгновенными сообщениями, в отличие от модулей для сайтов, не только набирают популярность, но и имеют значительно более широкие возможности для интеграции.

Второе множество решений состоит из различных систем для работы с клиентами, таких как CRM-системы и корпоративный чаты с различными интеграциями сервисов. Суть таких систем заключается в том, чтобы применить информацию накопленную за некоторое время работы с клиентами в каждом конкретном случае. Под накопленной информацией подразумеваются, например, данные о наличии различных товаров на складе, история заказов клиента, причастность клиентов к определенным социальным группам, и многое другое.

По мере развития сферы продаж, становится понятно, что объ-

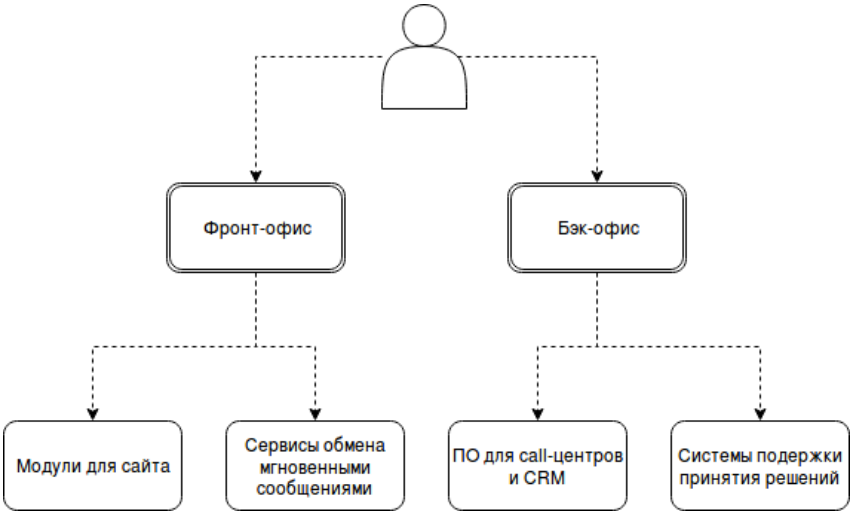

Рисунок 1.1. Общая схема решений для различных каналов взаимодействия с клиентами

единение удобного для клиентов формата общения в режиме реального времени в виде диалога со всей накопленной информацией со стороны сервиса увеличит эффективность консультаций и поднимет лояльность клиентов в десятки раз. Однако, возникает проблема, которая заключается в том, что во время живого общения с клиентом консультант физически не может обрабатывать все объемы информации в поисках ответа на запрос. К счастью, современные интеллектуальные системы при малых затратах на настройку могут сделать это за сотые доли секунд.

1.2. Классификация существующих программных средств

Роль онлайн-консультанта подразумевает за собой ряд обязанностей, в которые входит, помимо непосредственного консультирования, еще и воздействие на клиента, как на потенциального покупателя. Для этого любой консультант, в том числе и онлайн, должен иметь широкое представление о конкретном человеке, его интересах и воз-

можностях. Иными словами должна быть сформирована информационная база по каждому клиенту. Также большая часть диалогов в онлайн заканчивается телефонными звонками для уточнения подробностей. Исходя из всех вышеперечисленных аспектов работы, можно провести классификацию программных средств под каждый из них.

- Модули для сайта;
- Сервисы для обмена мгновенными сообщениями;
- ПО для call-центров;
- CRM-системы;
- Системы поддержки принятия решений.

Исходя из специфики данной работы, имеет смысл рассмотрение систем, которые в некой степени автоматизируют взаимодействие с клиентами, а также помогают в режиме реального времени принимать решения.

1.3. Обзор базовых средств

1.3.1. Модули для сайта

Сегодня подавляющее большинство интернет ресурсов, которые занимаются продажами имеют онлайн консультантов. Одной из самых крупных организаций по предоставлению онлайн консультантов является ресурс JivoSite. Цитата с сайта компании: "Чат – привычный способ общения в интернете, с мгновенным откликом и неформальный. Живое общение с посетителем сайта поможет ему сделать выбор, а вашему бизнесу увеличить продажи". Важная особенность JivoSite в том, что консультанты получают всю информацию, которую можно получить на сервере о клиенте. Сюда входит адрес в сети, с которого перешел пользователь на сайт, операционная система, браузер, история посещений курируемого сайта. Также есть возможность интеграции консультанта с CRM-системами, что позволяет еще больше расширить известную о пользователях информацию, а также систематизировано ее хранить. Сервис предоставляет возможности мониторинга чатов между консультантами и клиентами, получение подробной статистики за указанный промежуток времени, обратная связь от

клиентов получивших свою консультацию, рейтинги консультантов по результатам их работы.

Интеграция чата сводится к добавлению javascript кода и файлов разметки в структуру сайта. Со стороны консультантов сервис предоставляет веб, мобильные и десктопные приложения.

Однако, несмотря на все плюсы использования данного решения, оно активно использует человеческий ресурс. Другими словами, консультант получает на вход некий объем информации, но не всегда очевидно, как ее применить, к тому же, если опыт консультанта не так велик.

Помимо данного сервиса существует множество аналогичных решений. Например, Netrox SC, RedHelper, Cleversite и другие. Однако, с точки зрения процесса работы, они ничем не отличаются от рассмотренного.

1.3.2. Интеграция с сервисами для обмена мгновенными сообщениями

PandoraBots - открытая платформа для создания и последующей интеграции автоматических ботов с различными сервисами для обмена мгновенными сообщениями. Начнем с того, что PandoraBots имеют открытый и хорошо документированный API, что открывает горизонты для клиентов-разработчиков и развития сервиса в целом. Примечательно то, что API реализован сразу для трех языков программирования, а именно Java, Python и Javascript (Node.js). Более того, сообщество PandoraBots разработало также интерфейсы для Ruby, PHP и Go. Такой широкий выбор языка API безусловно только увеличивает потенциальную аудиторию среди разработчиков для сервиса.

Обучение ботов происходит на Artificial Intelligence Markup Language. AIML – язык разметки для искусственного интеллекта и является подмножеством (диалектом) XML.

Пример AIML файла:

```
<category>
  <pattern>WHAT IS YOUR NAME</pattern>
  <template><![CDATA[My name is <bot name="name"/>.]]></template>
</category>
<category>
  <pattern>WHAT ARE YOU CALLED</pattern>
  <template>
    <srai>what is your name</srai>
  </template>
</category>
```

Парадоксально то, что AIML, как удобное средство для обучения, является одновременно и сильной стороной сервиса, и слабой. Дело в том, что элементы этого языка разметки выстраивают однозначное соответствие между вопросом и ответом. Конечно значимость некоторых слов в вопросе можно опустить, а также прибегнуть к использованию переменных в составлении ответа, но это не изменяет того факта, что ответ будет получен только на тот вопрос, который уже есть в базе.

Конечно, даже при наличии подобных сервисов, большинство ботов разрабатывается программно с использованием API, но сам факт наличия подобных платформ говорит о степени развитости технологии.

Очевидно, что разработка средств для интеграции с сервисами для обмена мгновенными сообщениями не ограничивается только ботами, однако, с помощью них можно автоматизировать большинство аспектов данного канала связи. Более того, все флагманы рынка, а именно Whatsapp, Viber, и даже Skype, который изначально позиционировался, как видео-чат, с недавнего времени начали вести активную поддержку ботов в след за Telegram и Slack. Причем быстрый рост последних можно напрямую связывать с наличием платформы для поддержки ботов.

1.3.3. CRM-системы

SalesPlatform Vtiger CRM + Asterisk является примером свободно распространяемого программного продукта, который можно классифицировать, как CRM-систему с элементами ПО для call-центров. Система представляет из себя интеграцию Vtiger CRM и IP-АТС Asterisk.

Процесс работы с клиентом происходит следующим образом:

1. Звонок клиента автоматически перенаправляется в интерфейс CRM;

2. Если данному номеру клиента соответствует запись в системе, то звонок переводится на ответственного;

3. После принятия звонка в интерфейсе системы ответственным, появляется окно с данными из записи о звонящем. Предусмотрена возможность их редактирования во время разговора.

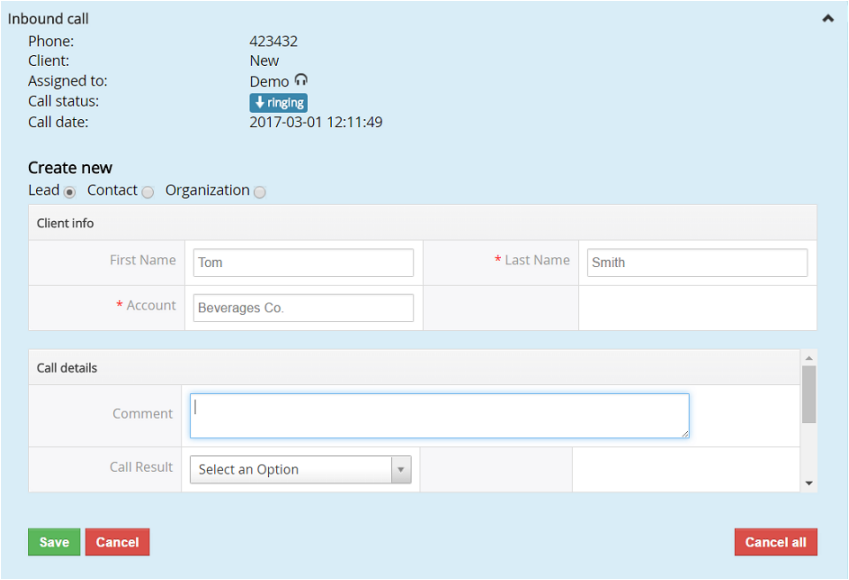

Рисунок 1.2. Интерфейс всплывающего окна SalesPlatform Vtiger CRM при звонке клиента

Также предусмотрен вывод поля, который будет содержать текстовый сценарий ведения диалога. Поле статичное и сценарии создаются для всех возможных ситуаций.

Подобный интерфейс присутствует в большинстве CRM с поддержкой IP-АТС.

1.3.4. Программное обеспечение для call-центров

Naumen Contact Center – полнофункциональное программное решение для построения крупных и средних контактных центров.

Одной из особенностей данного сервиса, по заявлению на сайте разработчика, являются подсказки оператору call-центра во время общения с клиентом. Однако, на деле эти подсказки выглядят, как набор клавиш с заголовками, которые описывают некоторые возможные тезисы со стороны клиента, например замечание о цене товара или услуги. При нажатии на такую клавишу, всплывает окно, которое,

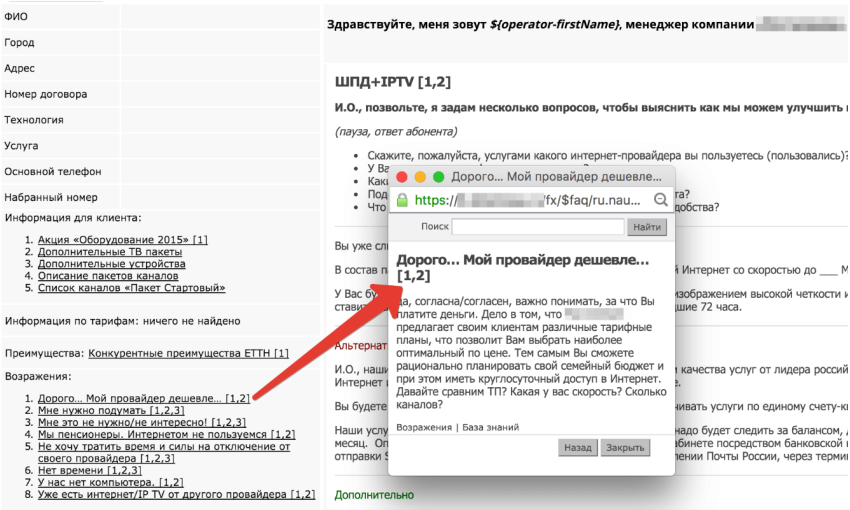

Рисунок 1.3. Интерфейс Naumen Contact Center

аналогично с решением SalesPlatform, содержит текстовое описание подходящих реплик.

1.4. Системы поддержки принятия решений

Система поддержки принятия решений — компьютерная автоматизированная система, целью которой является помощь людям, принимающим решение в сложных условиях для полного и объективного анализа предметной деятельности. Такие системы возникли в результате слияния управленческих информационных систем и систем управления базами данных.

Подобные системы не только позволяют повысить эффективность работы специалистов, но и увеличить процент принятия верных решений. Несмотря на все плюсы их использования, доля на рынке программного обеспечения для сферы продаж подобных систем стремится к нулю.

Исходя из всего вышеизложенного, мы имеем большой набор всевозможных каналов связи с клиентами, а также средства обработки информации о них. Симбиоз всех этих технологий может действитель-

но очень упростить и ускорить работу менеджеров и консультантов, однако, следующим шагом развития в этой сфере является внедрение искусственного интеллекта в систему, но, увы, полностью автоматические чат-боты пока не могут заменить человека. Поэтому можно сделать вывод, что наиболее перспективным вектором разработки являются интеллектуальные системы поддержки работы консультантов, которых на рынке в широком распространении нет.

1.5. Постановка задачи

Целью данного проекта является разработка интеллектуальной системы автоматизации рабочего места онлайн-консультанта, а также решение ряда задач:

- Создание системы по обработке запросов клиентов из сервиса обмена мгновенными сообщениями Telegram;
- Выбор и реализация одного из алгоритмов машинного обучения;
- Создание современного и гибкого интерфейса для визуализации графа диалога и взаимодействия с ним;
- Построение расширяемой архитектуры веб-приложения для последующего развития;
- Реализация взаимодействия клиентского приложения и серверного посредством протокола WebSocket для работы с приложением в режиме реального времени;
- Встраивание в архитектуру приложения базы данных для хранения информации о диалогах.

2. Проектирование архитектуры системы

2.1. Выбор средств разработки

2.1.1. Выбор сервиса для обмена мгновенными сообщениями

Как уже было сказано, разработка велась именно для Telegram. По данным сайта expandedramblings количество пользователей Telegram составило сто миллионов человек [1], в то время, как другим, набирающим популярность сервисом для обмена мгновенными сообщениями, Slack пользуются лишь пять миллионов [2]. Несмотря на такую разницу в размерах аудиторий, в планах на разработку также присутствует поддержка Slack.

Акцент сделан именно на этих программных продуктах, как на первых сервисах для обмена мгновенными сообщениями с поддержкой ботов, и, как следствие, имеющих подробную документацию и множество статей на тему разработки.

2.1.2. Средства разработки для backend

Разработка происходила на языке программирования Python для backend составляющей проекта. Подобный выбор был сделан исходя из желания сделать веб-приложение и модуль машинного обучения единообразными. Как известно, основными языками программирования для машинного обучения и анализа данных являются R и Python. Это обуславливаться их простотой и наличием большого количества специализированных библиотек. В то же время, для реализации серверной части веб-приложения хорошо подходят Java, PHP, Node.js, Python и многие другие. Пересекая два множества максимально подходящих языков, на пересечении мы получаем именно Python, который и был выбран для разработки.

Так как основной частью системы является веб-клиент, то важным пунктом в выборе средств разработки являлся выбор фреймворка для реализации веб-приложения. Для языка python имеется множество таких фреймворков, причем среди них есть "Full-Stack" (с базовым набором инструментов для клиентской составляющей проекта), такие как, Django, TurboGears, web2py, Giotto, Grok, и многие другие. А также "Non Full-Stack" , имеющие только инструменты для работы

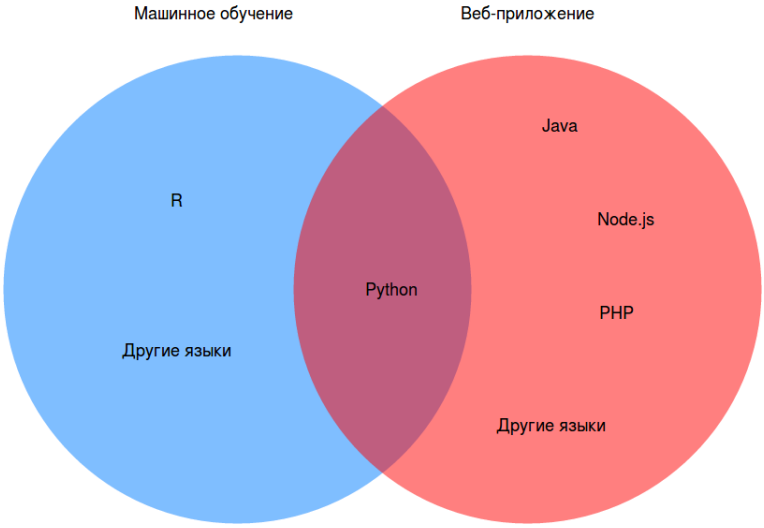

Рисунок 2.1. Языки программирования для машинного обучения и веб-приложений

с серверной частью. Остальные модули либо пишутся, либо устанавливаются в виде библиотек вручную. Среди них Flask, CherryPy, Bottle, Pyramid, Hug.

Выбор производился исходя из того факта, что клиентская часть будет строиться вокруг визуализатора json объектов. Иными словами, готовое решение в рамках веб-фрейморка для frontend составляющей не было необходимо. Также далеко не все "Full-Stack" фреймворки имеют встроенную поддержку базы данных Redis. Поэтом было принято решение выбирать только среди "Non Full-Stack" фрейморков, настройка которых также является полезным опытом в плане изучения структуры классического веб-приложения.

Далее, из всех перечисленных вариантов были выбраны те, которые имеют наиболее широкое распространение, а также имеют поддержку python 3. Таким образом остались Flask, CherryPy и Pyramid. Далее были рассмотрены документации всех представленных решений и выбор пал именно на Flask, который имеет не только хорошо проработанную документацию, но и множество авторский статей на

тему разработки веб-приложений, которые посвящены именно этому фреймворку.

Следующим пунктом в выборе средств разработки являлся выбор библиотеки для машинного обучения. Была выбрана библиотека scikit-learn, как самый развитый пакет для этого на языке python, а также имеющая подробную документацию.

Аналогично выбор морфологического анализатора свелся к единственном варианту pymorphy2 [3]. Так как из всего многообразия анализаторов только этот имеет поддержку русского языка и подробную документацию.

Библиотека TeleBot была выбрана для использования Telegram Bot API, так как вторая по величине библиотека - Telepot, имеет не такую подробную документацию и все руководство сводится к ограниченном набору примеров использования.

Так как для реализация веб-сокетов на клиентской части происходит с помощью библиотеки SocketIO, то логично будет использовать ее также и для серверной. Библиотека Flask-SocketIO применяется для real-time связи между сервером и клиентом. Данная библиотека является инструментом для использования вместе с Flask библиотеки SocketIO - флагмана среди фреймворков для работы с вебсокетами.

2.1.3. Средства разработки для frontend

Для создания frontend составляющей проекта использовался следующий набор технологий:

- Язык программирования Javascript;
- Визуализатор json объектов Jsonschemaviewer;
- Frontend фреймворк Twitter Bootstrap;
- SocketIO для real-time связи между сервером и клиентом.

2.2. Выбор базы данных

В ходе разработки стало очевидно, что необходимо сделать возможным расширение базы вопросов и ответов в процессе работы. Причем это расширение должно происходить автоматизированно и быстро, чтобы свести временной промежуток между тем, когда клиент

задаст вопрос, и тем, когда же составленный граф отобразится у консультанта, к нулю. Таким образом, появилась необходимость в базе данных, которая имела бы наиболее высокую производительность.

Redis - высокопроизводительная нереляционная база данных, возможности которой позволяют выполнить 1000000 SET запросов (запись в базу) за 13.86 секунд, как говорят данные на сайте разработчика. По основным принципам Redis схожа с другой нереляционой базой MemcacheDB, однако, имеет важное отличие, которое и склоняет выбор именно к ней, а именно, MemcacheDB конечно позволяет хранить массивы, но перед этим преобразует их и сохраняет в виде строк, что делает невозможным проводить атомарные операции с такими записями. Конечно существует подвид этой базы, который решает эту проблем, а именно Memcached, однако, в связи с более широким распространением Redis и тем, что при кешировании она показывает лучшие результаты [4], решено было выбрать именно эту базу данных.

Хранение информации о вопросах и ответах на них производится в виде массивов с сообщениями из диалога. Во-первых, это позволяет удобно получать нужные элементы записи, а во-вторых, используя подобный способ хранения, можно записывать диалоги различной длины не ограничиваясь количеством полей или ключами в хэш-таблице.

Каждая таблица имеет обозначение вида "qa_X" где "X" - порядковый номер записи.

2.3. Взаимодействие сервера и клиента через Websocket

Сложно представить современные веб-приложения без вебсокетов, когда находясь на одном интернет ресурсе, можно часами не обновлять интернет страницу, однако, информация на ней будет постоянно обновляться. Так как основной целью разработки было создание приложения для работы в режиме реальном времени, то невозможно не прибегнуть именно к этой технологии.

Связь клиента и сервера происходит посредством передачи сообщений через вебсокеты. Существуют следующие виды сообщений:

1. "Connect". Приходит от клиента к серверу в момент подключения первого. Не содержит данных;

2. "Connect response". Отправляется с сервера клиенту, когда первый зафиксировал подключение. Содержит строку "Connected";

3. "Update json". Пакет с данным событием передает сообщение о том, что клиент задал вопрос. В пакете передается сам вопрос, а также предполагаемые варианты ответа от сервера. Передается от сервера оператору-консультанту;

4. "Receive answer". Пакет с данным событием передает сообщение о том, что оператор выбрал подходящий ответ. В пакете передается выбранный ответ;

5. "Finish dialog". Пакет с данным событием передает сообщение о том, что клиент завершил диалог;

6. "Disconnect". Приходит от клиента к серверу в момент отключения первого. Не содержит данных.

Обмен пакетами происходит через выделенное пространство имен "/socket"для того, чтобы не зашумлять общее пространство имен.

2.4. Принципы построения системы автоматизации работы онлайн-консультанта

2.4.1. Анализ доступных алгоритмов машинного обучения

Выбор подходящего алгоритма машинного обучения является важным пунктом в разработке любой системы предсказания и прогнозирования. Этот выбор в большинстве случаев зависит от специфики анализируемых данных, а также от условий анализа.

Все алгоритмы машинного обучения делятся на три типа: контролируемые, неконтролируемые и подкрепляемое обучение. Контролируемое обучение применяется в тех случаях, когда имеется массив данных, у элементов которого имеются определенные явные свойства, но для другого массива эти свойства неявны и требуется их предсказать в ходе работы. Неконтролируемое обучение подходит для выявления неявных отношений в заданном не размеченном наборе данных. Подкрепляемое обучение является симбиозом вышеописанных категорий, и его суть заключается в наличии некоторого вида обратной связи, которая доступна для каждого шага или действия, но отсутствует

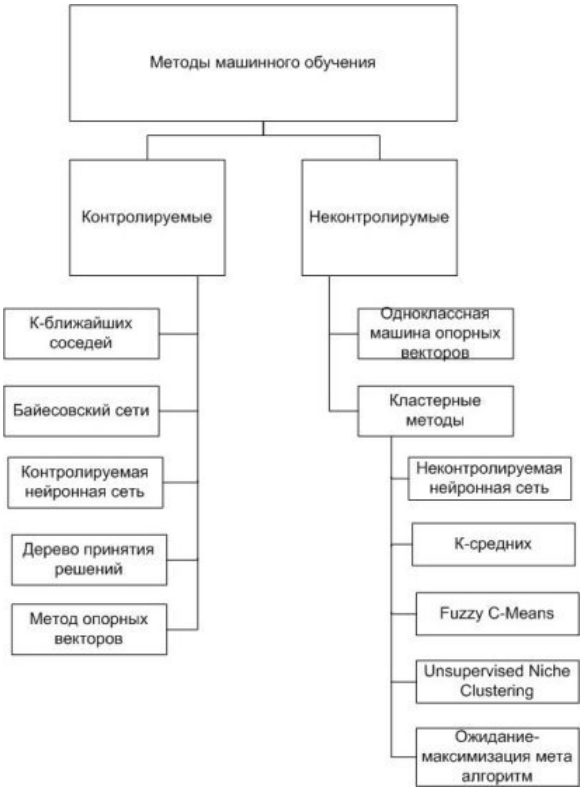

Рисунок 2.2. Схема некоторых алгоритмов машинного обучения

какая-либо разметка и сообщение об ошибке. В случае с анализом вопроса, имеется база вопросов и ответов, а также явные признаки в виде наличия тех или иных слов в вопросе. В связи с этим, принято решение использовать один из контролируемых алгоритмов.

Далее выделим основные задачи, которые могут решать подобные алгоритмы

- Классификация. Данная задача заключается в определении принадлежности исследуемого объекта к тому или иному классу, подмножеству. В случае, если определение происходит только при наличии двух вариантов, то подобная классификация на-

зывается биномиальной или двухклассной. Если же категорий, как, например, при поиске среди множества подобных вопросов, много то, классификация называется многоклассовой;

- Регрессия. Эта задача базируется на предсказании степени выраженности того или иного признака у заданного набора данных. К таким задачам относится, например, предсказание стоимости жилья при наличии информации о его месте нахождения;

- Обнаружение аномалий. В таких случаях задача состоит в выявлении точек данных, которые выбиваются из общего набора по некоторым признакам. Например, при идентификации мошенничества с банковской картой подозрительными являются любые операции оплаты, которые имеют слишком разнящиеся признаки, будь то место снятия денег, время или сумма транзакции.

В данной работе выбор стоял между задачей классификации, а именно, задать каждый вопрос из базы, как отдельный класс, и между задачей регрессии, в частности, можно разбить вопросы по тематикам и определять вопрос по степени причастности к каждой из тематик. Очевидно, что именно классификация является оптимальным решением, поскольку решение задачи регрессии в текущем проекте потребует слишком масштабную предобработку данных, а также дает далеко не самые точные результаты, так как многие вопросы могут иметь идентичную причастность к заданным тематикам.

Следующий этап заключается в выборе подходящего алгоритма многоклассовой классификации, и главную роль при этом сыграл тот факт, что в качестве признаков вопроса выступает наличие тех или иных слов. Важно то, чтобы более редкие и значимые слова играли ключевою роль в классификации, а менее значимые слова использовались бы позже. Подобная модель практически полностью соответствует алгоритму решающего дерева, у которого в основании лежат признаки с наибольшими весами, а ближе к концам ветвей признаки с наименьшими весами.

В одном из первых вариантов проекта этот алгоритм и был взял за основу, однако, сразу же выяснилось, что качество прогнозов очень низкое. Это связано, во-первых, с тем, что множество не значимых слов в вопросе создает большую зашумленность выборки, во-вторых, небольшие масштабы обучающей выборки приводят к явлению называемому переобучаемостью. Данное определение подразумевает под

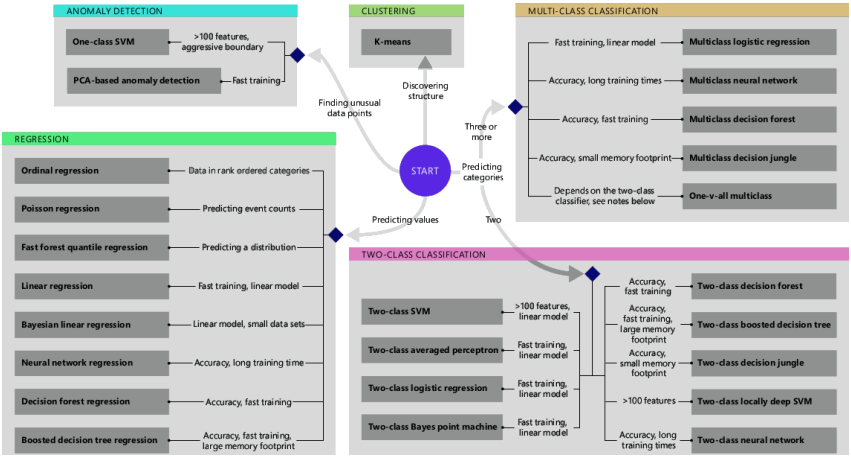

Рисунок 2.3. Памятка по алгоритмам Машинного обучения Microsoft[5]

собой процесс подстраивания модели для прогнозов под обучаемую выборку, но получение неудовлетворительных результатов на тестовой выборке. В связи с этим, решающее дерево было заменено на случайный лес, который является эволюцией предыдущего алгоритма. Его принцип заключается в объединении в один ансамбль нескольких обученных случайных деревьев, которые, в последствие, производят голосование, и по результатам этого голосования случайный лес подготавливает ответ по заданной выборке. В нашем случае лес классифицирует заданный вопрос по классам вопросов из обучающей выборки, а также подсчитывает вероятность причастности вопроса к каждому из классов.

В качестве подкрепления выбора была рассмотрена памятка-граф разработанная компанией Microsoft для выбора алгоритма машинного обучение для различных задач[5].

Из графа можно сделать вывод, что для задач многоклассовой классификации, когда в приоритете стоят точность и скорость обучения, наиболее подходящим является метод случайного леса.

2.4.2. Точность предлагаемых ответов

Также было произведено тестирование алгоритма при обучающей выборке в 250 записей и тестовой в 50 записей. Тестовая выборка составлялась путем перефразирования вопросов так, чтобы их логический смысл оставался таким же. Количество правильных ответов составило 72%, причем, если учитывать то, что оператору предоставляется три наиболее вероятных ответа, то среди них частота встречи правильного ответа составила 88%.

2.4.3. Процесс обучения в разрабатываемом проекте

Процесс обучения состоит из нескольких этапов.

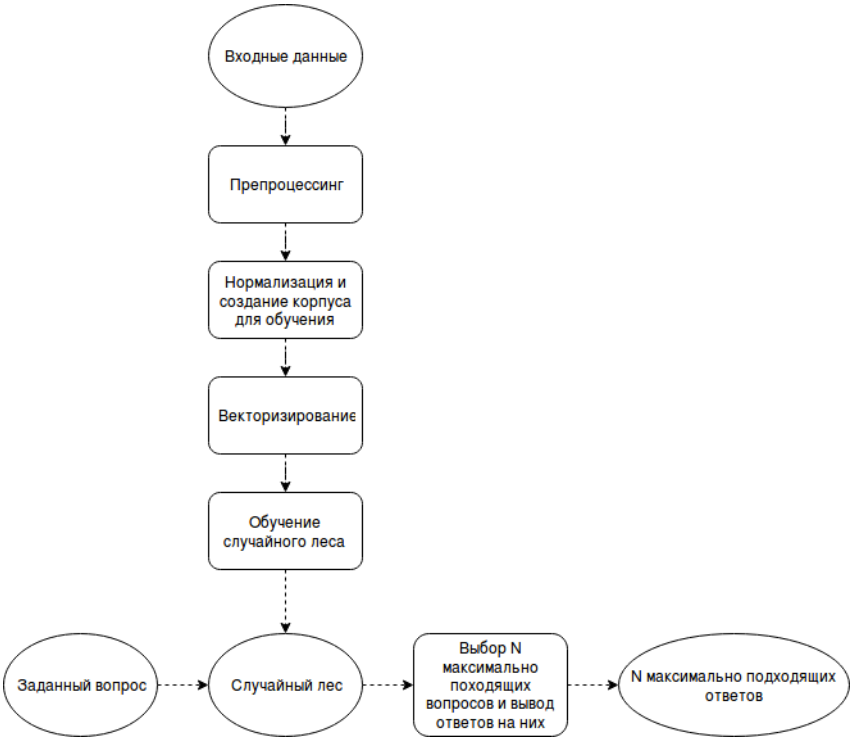

Рисунок 2.4. Общая схема процесса машинного обучения

Во-первых, происходит так называемый препроцессинг базы диалогов и их возможного продолжения, после которого остаются только те наборы, которые соответствуют текущему этапу диалога.

Во-вторых, из подготовленного набора вопросов составляется корпус для обучения. Перед этим текст каждого вопроса приводится к единой форме, а именно, из них удаляются символы, а также слова написанные на английском языке заменяются на их транслитерацию на русском языке. Это сделано для того, чтобы корректно обрабатывать некоторые англоязычные термины, которые пользователи могут написать на русском языке. Например, вместо "CRM"часто используют "СРМ". Также все слова нормализуются средствами морфологического анализатора pymorphy2, таким образом, каждое слово приводится к своей начальной форме, при том причастия обращаются в глагольный инфинитив. Это позволяет одинаково обрабатывать одни и те же слова в разных контекстах.

В-третьих, из составленной и нормализованной выборки вопросов создается TF-IDF (Term Frequency / Inverse Document Frequency) векторизатор. Данный процесс подразумевает под собой вычисление относительных весов каждого слова в рамках всей выборки. Вес слова высчитывается делением частоты встречи слова на частоту его встречи в других вопросах. Таким образом, слова, которые обозначают интересующие пользователя услуги, технологии, сферы деятельности имеют веса намного большие, чем слова составляющие остальную часть вопроса.

Затем, на базе подготовленного векторизатора производится обработка всей базы вопросов и обучение случайного леса.

В момент, когда консультант выбирает подходящий ответ, он сразу же помещается в базу вместе с вектором текущего диалога. Таким образом, система обучается в процессе работы.

3. Разработка системы

3.1. Функциональность

В состав приложения входят веб-клиент и Telegram бот. Основные идеи, которые были реализованы в приложении представлены далее:

- Обработка сообщений пользователей в режиме реального времени;
- Мгновенное представление возможных вариантов ответа консультанту;
- Обработка решения консультанта в режиме реального времени и пересылка ответа клиенту;
- Динамическое расширение базы диалогов во время работы системы.

Веб-клиент, в свою очередь состоит из интерфейса для работы с графом диалога, а также клавишей для отправки сообщения. Со стороны клиента общение сводится к классическому сценарию работы с Telegram ботом.

3.2. Описание элементов проекта

При включении проекта запускаются два потока. Один поддерживает работу Telegram бота, а второй веб-приложения. Информация между потоками передается посредством двух переменных: "answer_array" которая является массивом и содержит сообщения пользователя и оператора в текущем диалоге, а также "data" которая является словарем и хранит текущее состояние диалога в виде дерева.

3.2.1. Поток для работы с ботом

Обработка сообщений клиента происходит с помощью средств "message_handler" библиотеки TeleBot.

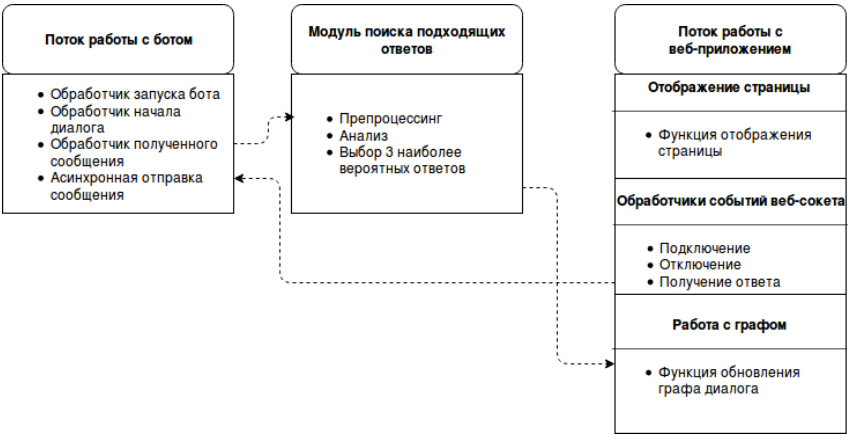

Рисунок 3.1. Общая схема проекта

Листинг 3.1. Фрагмент программного кода с обработкой сообщений от клиента

```
@bot.message_handler(func=lambda message: message.text != '',
     content_types=['text'])
def handle_text(message):
    """Функция обработчика сообщения

    Keyword arguments:
    message — объект сообщения

    """
    answer_array.append(message.text)
    answer = """Пожалуйста, подождите."""
    bot.send_message(message.chat.id, answer)

    log(message, answer)

    update_json(answer_array, message.chat.id)
```

Такие обработчики позволяют отслеживать сообщения от клиента и по определенным условиям производить соответствующие действия.

3.2.2. Поток для работы веб-приложения

В этом потоке происходит запуск Flask приложения. Был реализован ряд функций для обработки событий веб-сокетов, работы с базой данных, а также отображения стартовой страницы.

Листинг 3.2. Функция для отображения стартовой страницы

```
@app.route('/viewer/')
def get_view():
    get_view.data = {
        "title": "Ожидание диалога"
    }
    return render_template('index.html',
                           context=json.dumps(get_view.data,
                               separators=(', ', ': ')))
```

В ходе разработки выяснилось, что визуализатор json объектов использует формат, который крайне неудобно обрабатывать и расширять.

Листинг 3.3. Пример json объекта в формате поддерживаемом визуализатором

```
{'title': 'Здравствуйте, мне нужна помощь.',
              'properties':
                  [ {
                  'title': 'Здравствуйте, какого рода помощь вам тре
                      бется?', 'properties': []
                  }, {
                  'title': 'Здравствуйте, с удовольствием вам поможе
                      м.', 'properties': []
                  }, {
                  'title': 'Добрый день. Опишите подробнее вашу проб
                      лему.', 'properties': []
                  } ]
}
```

Так как в процессе работы наилучшим способом хранения текущего состояния диалога является массив, очевидно, что использовать подобный формат для динамического преобразования крайне неудобно. Поэтом было принято решение конвертировать json объект для расширения в другой вид.

Листинг 3.4. Пример json объекта в формате подходящем для преобразований

```
{'Здравствуйте, мне нужна помощь.': {
    'Здравствуйте, какого рода помощь вам требуется?': {},

    'Здравствуйте, с удовольствием вам поможем.': {},

    'Добрый день. Опишите подробнее вашу проблему.': {},
}
}
```

Такой формат позволяет обращаться к каждой ветви дерева диалога зная предшествующие значения следующим образом:

tree_object[first][second][third][etc]. Причем количество промежуточных ветвей в этом случае неважно.

Исходя из этого, были разработаны функции для конветации json объектов из подробного формата в компактный и наоборот.

Листинг 3.5. Функции для конвертирования json объектов

```
def verbose_to_compact(verbose):
    """Функция для преобразования подробного представления json в ко
        мпактное

    Keyword arguments:
    verbose — подробный json

    """
    return { item['title']: verbose_to_compact(item['properties'])
        for item in verbose }


def compact_to_verbose(compact):
    """Функция для преобразования компактного представления json в п
        одробное

    Keyword arguments:
    compact — компактный json

    """
    return [{'title': key, 'properties': compact_to_verbose(value)}
        for key, value in compact.items()]
```

Также появилась необходимость не только получать значения в ветвях дерева зная промежуточные значения, но и изменять их. Для этих целей также были написаны функции.

Листинг 3.6. Функции для получения и изменения значения ветви дерева по массиву значений в промежуточных узлах

```
def getFromDict(dataDict, mapList):
    """Функция получения ветви в json по массиву значений в каждом у
        зле

    Keyword arguments:
    dataDict — json дерево
    mapList — массив значений

    """
    return reduce(operator.getitem, mapList, dataDict)


def setInDict(dataDict, mapList, value):
    """Функция задания значения ветви в json по массиву значений в к
        аждом узле

    Keyword arguments:
```

```
    dataDict — json дерево
    mapList — массив значений
    value — новое значение

    """
    getFromDict(dataDict, mapList[:-1])[mapList[-1]] = value
```

Помимо этого, в данном потоке используется функция, которая обрабатывает текущий вектор диалога и на его основе возвращает вероятные варианты ответа. Данная функция является препроцессором и декоратором для функций анализа.

Листинг 3.7. Препроцессинг текущего вектора диалога

```
def preprocessing(answer_array):
    """Функция, которая подготавливает выборку в зависимости текущег
        о состояния массива диалога

    Keyword arguments:
    answer_array — Вектор сообщений в текущем диалоге

    """

    r_server = redis.StrictRedis('localhost', charset="utf-8",
        decode_responses=True)

    qa = []
    for key in r_server.scan_iter():
        row = r_server.lrange(key, 0, r_server.llen(key))
        row.reverse()

        if len(answer_array) == 1:
            if len(row) == 2:
                qa.append(Question(row[-2], row[-1]))
        else:
            if answer_array[0:-1] == row[0:-2]:
                qa.append(Question(row[-2], row[-1]))

    if not qa:
        return []
    else:
        return analyze(qa, answer_array[-1])
```

Интерфейс приложения является ответвлением проекта jsonschemaviewer. В ходе разработки для данного проекта была реализована связь с серверной частью через веб-сокет, обработка нажатия на нужное сообщение, а также его отправка. Помимо этого, добавлена возможность отправлять собственные сообщения из окна отображения графа.

Листинг 3.8. Инициализация веб-сокета на клиентской стороне и обработка нажатия на клавишу отправки

```
1   var socket = io.connect('http://localhost:' + location.port + '/
        socket');
2   socket.on('update json', function(json_obj) {
3       visualizeView(json_obj);
4   });
5
6   function setupEvents() {
7       $('#sendButton').click(sendToServer);
8   };
9
10  function sendToServer() {
11      socket.emit('receive answer', $("#txt-for-send").val());
12      $("#txt-for-send").val('');
13  };
```

4. Тестирование системы

Тестирование системы производилось вручную. После каждой конечной итерации разработки выполнялась проверка работоспособности по заранее подготовленным тест-кейсам, которые были составлены исходя из целей разработки и намеченных инструментов.

1. Работа с Telegram ботом:
    - Отклик бота на сообщение о начале работы;
    - Корректная обработка неожиданных сообщений от пользователя.
2. Веб-приложение:
    - Корректное отображение графа диалога;
    - Обработка нажатия на клавишу отправки сообщения;
    - Скорость отображения графа после того, как клиент; задал вопрос;
    - Обновление базы данных после ответа консультанта.
3. Машинное обучение:
    - Получаемые ответы на вопрос совпадающий с некоторым вопросом из базы;
    - Получаемые ответы на вопрос, который является морфологически измененным вопросом из базы;
    - Получаемые ответы на вопрос, который является отличным от любого вопроса из базы, но имеющим тот же логический смысл.

4.1. Тестирование процесса диалога со стороны клиента

Со стороны клиента ведение диалога выглядит следующим образом. Сначала пользователь добавляет бота к себе в список контактов. Далее нужно запустить его, как и любого другого бота Telegram, командой "/start" .

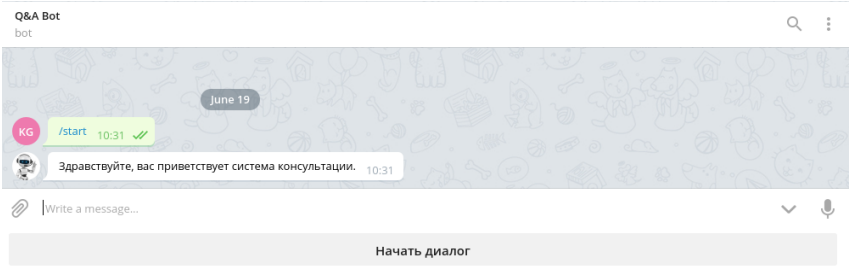

Рисунок 4.1. Реакция бота на запуск

После приветственного сообщения пользователю нужно ввести "Начать диалог"или воспользоваться уже подготовленной для этого клавишей в интерфейсе приложения для обмена мгновенными сообщениями.

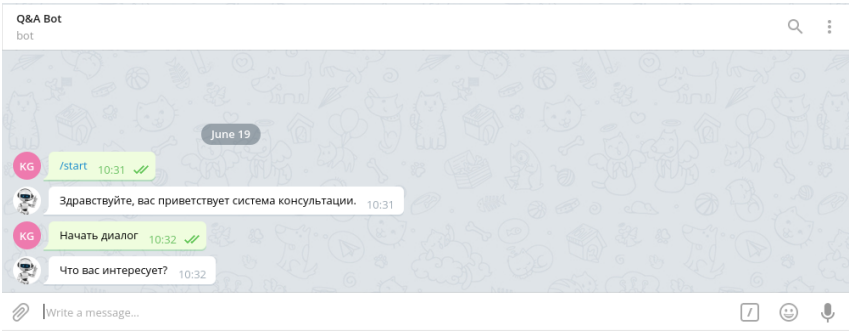

Рисунок 4.2. Предложение задать вопрос

Далее пользователь должен ввести интересующий его вопрос, проблему или сервис, с которым возникли проблемы.

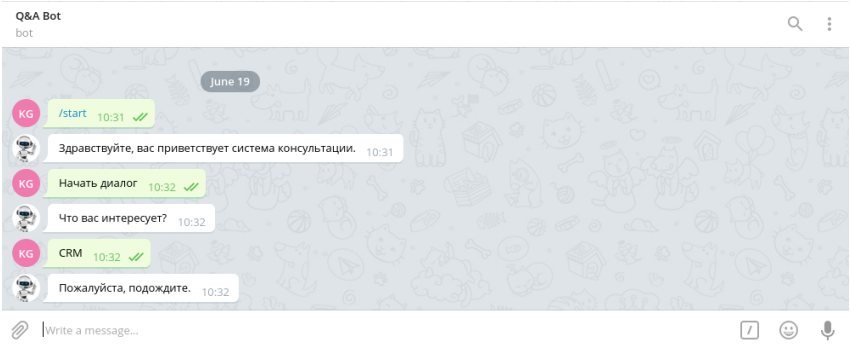

Рисунок 4.3. Обработка сообщения

Затем бот получает вопрос пользователя, отправляет просьбу об ожидании и высылает вопрос на обработку оператору-консультанту. После выбора оператором подходящего ответа, текст этого ответа приходит пользователю в виде сообщения от бота.

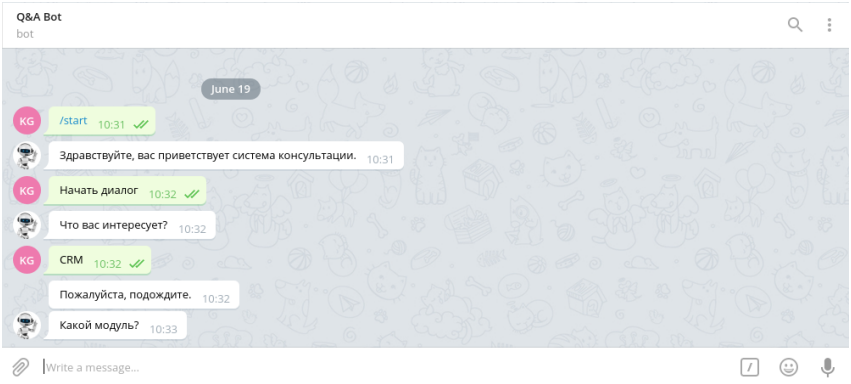

Рисунок 4.4. Сообщение от оператора

Далее диалог ведется по аналогичной схеме, до тех пор, пока пользователь сам его не закончит.

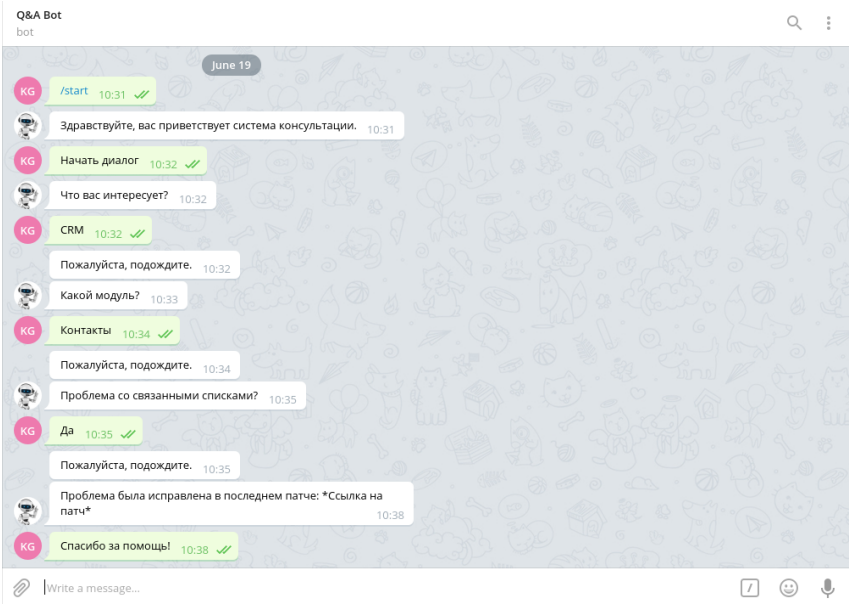

Рисунок 4.5. Окончание диалога

4.2. Тестирование процесса диалога со стороны консультанта

Для начала работы консультант включает приложение, тем самым переводя его в режим ожидания.

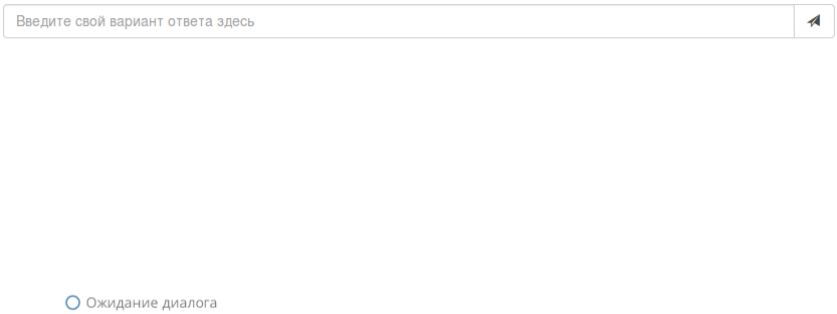

Рисунок 4.6. Ожидание начала работы

После того, как клиент задаст вопрос и он будет обработан системой, в интерфейсе консультанта появляется граф диалога.

Рисунок 4.7. Общий вид графа диалога

Во главе графа находится вопрос пользователя и возможные ответы. При клике на один из ответов, его текст автоматически попадает

в графу ответа.

Рисунок 4.8. Выбор подходящего ответа

Далее снова ожидается ответ клиента на отправленное сообщение. После получения этого ответа граф перестраивается, добавляя к ветви с выбранным ответом сообщение пользователя и новые подходящие ответы подготовленные интеллектуальной системой.

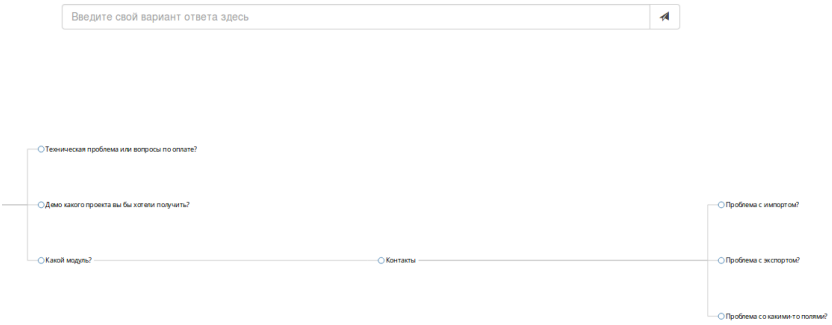

Рисунок 4.9. Продолжение ветви графа диалога

Если консультант посчитает, что предложенные ответы не подходят в данной ситуации, то он всегда может ответить сам с помощью формы отправки сообщения.

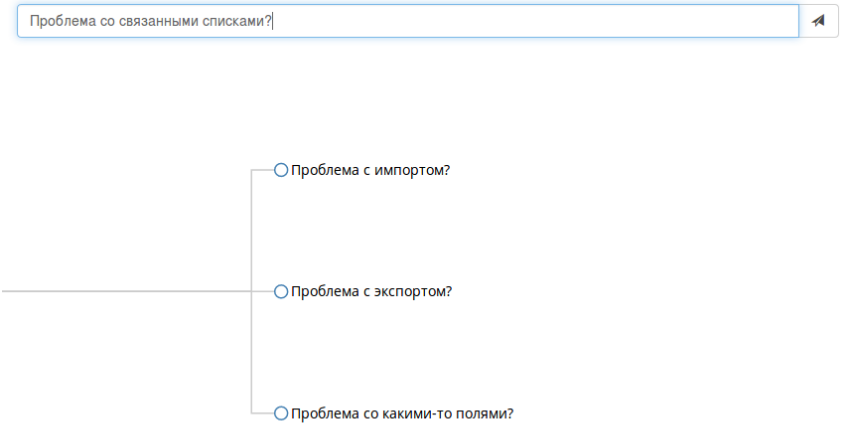

Рисунок 4.10. Самостоятельный ответ оператора

В этом случае система добавляет новую ветвь развития диалога и записывает такой сценарий в базу данных, чтобы в следующий раз

при обучении был обработан и этот вариант.

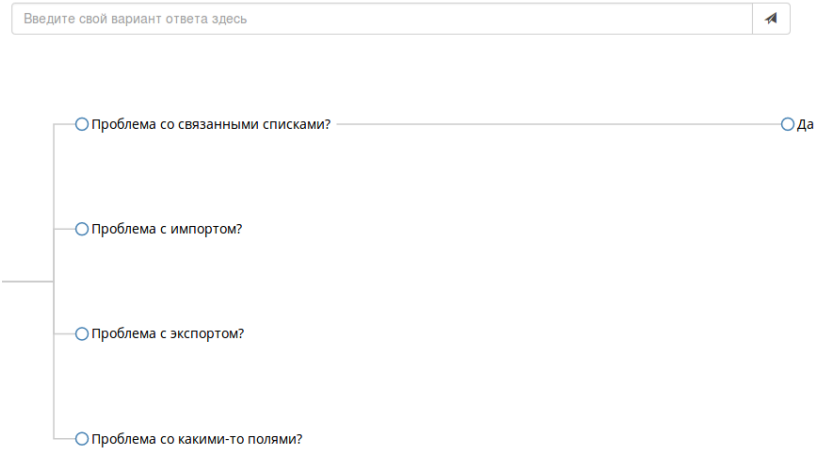

Рисунок 4.11. Создание новой ветви и ее продолжение

ЗАКЛЮЧЕНИЕ

В результате выполнения работы были исследованы различные технологии для автоматизации рабочего места специалистов по работе с клиентами, в частности, онлайн-консультантов. По результатам исследования можно сказать, что даже при быстром современном развитии интеллектуальных систем, их роль в программном обеспечении для консультирования сводится к нулю. Отчасти это связано с тем, что консультирование необходимо производить в режиме реального времени, а большая часть алгоритмов для машинного обучения требует большое количество времени на препроцессинг и само обучение.

Таким образом, целью работы было создание веб-приложения для онлайн-консультантов, анализирующая часть которой, могла бы предлагать возможные пути развития диалога в режиме реального времени. Более того, нужно было предусмотреть возможность дообучения в процессе работы для повышения качестве предлагаемых ответов, а также для обработки большого количества частных ситуаций.

По итогам разработки цель была полностью достигнута. И, как результат, конечным продуктом стало полноценное веб-приложение для онлайн-консультантов, которое имеет высокую скорость отклика и наглядную визуализацию процесса диалога.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. 10 Interesting Telegram stats [Электронный ресурс], expandedramblings. — URL: expandedramblings.com/index.php/telegram-stats/ (дата обращения: 20.04.2017).
2. 40 Interesting Slack stats [Электронный ресурс], expandedramblings. — URL: http://expandedramblings.com/index.php/slack-statistics/ (дата обращения: 20.04.2017).
3. Korobov Mikhail. Morphological Analyzer and Generator for Russian and Ukrainian Languages // Analysis of Images, Social Networks and Texts / Ed. by Mikhail Yu. Khachay, Natalia Konstantinova, Alexander Panchenko et al. — Springer International Publishing, 2015. — Vol. 542 of Communications in Computer and Information Science. — P. 320–332. — URL: http://dx.doi.org/10.1007/978-3-319-26123-2_31.
4. Why Redis beats Memcached for caching [Электронный ресурс], Infoworld. — URL: http://www.infoworld.com/article/3063161/application-development/why-redis-beats-memcached-for-caching.html (дата обращения: 25.05.2017).
5. Machine learning algorithm cheat sheet for Microsoft Azure Machine Learning Studio [Электронный ресурс], Microsoft. — URL: https://docs.microsoft.com/en-us/azure/machine-learning/machine-learning-algorithm-cheat-sheet (дата обращения: 25.05.2017).

ПРИЛОЖЕНИЕ А
Техническое задание

Веб-приложение - интеллектуальная система автоматизации работы онлайн-консультанта.

Техническое задание.

1 лист

Введение

Наименование программы: «Интеллектуальная система автоматизации работы онлайн-консультанта». Данное средство предназначено для автоматизации рабочего места онлайн-консультанта.

Назначение разработки

Разработка системы поддержки принятия решений для онлайн-консультантов с графическим интерфейсом визуализации графа диалога.

Требование к программе или программному изделию

Требования к функциональным характеристикам

Разработанное приложение должно предоставлять веб-интерфейс для общения с клиентами, а также возможные варианты ответов, в зависимости от сообщений.

Требования к составу и параметрам технических средств

Разработанное веб-приложение может функционировать на всех популярных веб-браузерах, в том числе, и на их мобильных аналогах.

Требования к информационной и программной совместимости

Серверная часть программы должна быть написана на языке Python с использованием фреймворка Flask, библиотеки scikit-learn и базы данных Redis. Клиентская часть программы должна быть написана на языке Javascript.

ПРИЛОЖЕНИЕ Б
Описание программы

Веб-приложение - интеллектуальная система автоматизации работы онлайн-консультанта.

Описание программы.

1 лист

Общие сведения

Разработанное приложение предоставляет веб-интерфейс для общения с клиентами, а также возможные варианты ответов, в зависимости от сообщений.

Используемые технические средства

Для функционирования приложения, необходимо подключение к сети Internet, наличие токена зарегистрированного Telegram бота, а также любой доступный веб-браузер.

Входные данные

Входными данными является токен зарегистрированного Telegram бота.

ПРИЛОЖЕНИЕ В
Программа и методика испытаний

Веб-приложение - интеллектуальная система автоматизации работы онлайн-консультанта.

Программа и методика испытаний.

2 листа

Объект испытаний

Наименование программы – объекта испытаний: «Интеллектуальная система автоматизации работы онлайн-консультанта».

Цель испытаний

Целью проведения испытаний является проверка соответствия функциональных характеристик средства требованиям, указанным в техническом задании.

Требования к программе

Разработанное средство должно функционировать в веб-браузере, обрабатывать сообщения пользователей и, если это возможно, предлагать вероятные варианты ответов. Также должна производиться запись в базу Redis в процессе работы.

Средства и порядок испытаний

Программа должна выполняться веб-браузере. Для проведения испытаний был собран тестовый стенд, состоящий из компьютера и мобильного устройства, находящихся в одной сети.

Компьютер с Flask сервером, базой данных и клиентом в веб-браузере:

1. Оборудование:
    - Intel Core i7-3632QM CPU @ 2.20GHz x 4;
    - Оперативная память – 8 Гб;
    - Жёсткий диск – 500 Гб;
    - Карта сетевого интерфейса(NIC);
    - Стандартный монитор;
    - Клавиатура;
    - Мышь.
2. Операционная система:
    - Linux Mint 18.1 Serena
3. Установленное ПО:
    - Python 3.5.2;
    - Фреймворк Flask 0.12.1;
    - Сервер базы данных Redis 3.2.9;

- Веб-браузер Mozilla Firfox 54.0.

Мобильное устройство с приложением Telegram:

1. Оборудование:
   - Just5 Blaster Mini;
   - Сим-карта с подключением к сети Internet;
2. Операционная система:
   - Android 5.0.
3. Установленное ПО:
   - Telegram для Android;

Порядок проведения испытаний:

1. запуск сервера Redis;
2. запуск сервера Flask;
3. проверка на наличие подключениях к сети Internet все устройств;
4. запуск веб-приложения;
5. запуск приложения Telegram;
6. добавление бота в список контактов;
7. проверка на соответствие функциональных характеристик, указанных в техническом задании;
8. анализ проведённых испытаний.

ПРИЛОЖЕНИЕ Г
Листинги

Веб-приложение - интеллектуальная система автоматизации работы онлайн-консультанта.

Фрагменты кода программы.

8 листов

Листинг Г.1. run.py

```
1  from flask import Flask, render_template, json
2  from flask_socketio import SocketIO, emit
3  from main import preprocessing
4  from telebot import types, TeleBot
5  from telebot.util import async
6  import _thread
7  import redis
8  from functools import reduce
9  import operator
10 import config
11
12 # Создание объекта приложения Flask
13 app = Flask(__name__)
14
15 # Создание объекта SocketIO
16 socketio = SocketIO(app)
17
18 # Создание объекта TeleBot
19 bot = TeleBot(config.token)
20
21 # Массив для хранения сообщений диалога
22 answer_array = []
23
24 # Словарь для хранения json представления графа диалога
25 data = {}
26
27
28 def getFromDict(dataDict, mapList):
29     """Функция получения ветви в json по массиву значений в каждом у
           зле
30
31     Keyword arguments:
32     dataDict — json дерево
33     mapList — массив значений
34
35     """
36     return reduce(operator.getitem, mapList, dataDict)
37
38
39 def setInDict(dataDict, mapList, value):
40     """Функция задания значения ветви в json по массиву значений в к
           ждом узле
41
42     Keyword arguments:
43     dataDict — json дерево
44     mapList — массив значений
45     value — новое значение
46
47     """
48     getFromDict(dataDict, mapList[:-1])[mapList[-1]] = value
49
50
51 def verbose_to_compact(verbose):
52     """Функция для преобразования подробного представления json в ко
           мпактное
53
```

```
54      Keyword arguments:
55      verbose — подробный json
56
57      """
58      return { item['title']: verbose_to_compact(item['properties'])
            for item in verbose }
59
60
61  def compact_to_verbose(compact):
62      """Функция для преобразования компактного представления json в п
            одробное
63
64      Keyword arguments:
65      compact — компактный json
66
67      """
68      return [{'title': key, 'properties': compact_to_verbose(value)}
            for key, value in compact.items()]
69
70
71  def update_json(answers, client_id):
72      """Функция для обновления объекта графа и id клиента
73
74      Keyword arguments:
75      answers — массив диалога
76      client_id — id клиента
77
78      """
79      global current_chat_id
80      current_chat_id = client_id
81      result = preprocessing(answers)
82      branch = {'title': answers[-1],
83                'properties':
84                    [{'title': key, 'properties': []} for key in
                          result]
85                }
86
87      if len(answer_array) == 1:
88          get_view.data = branch
89      else:
90          new = verbose_to_compact([get_view.data])
91          new_branch = verbose_to_compact([branch])
92          setInDict(new, answer_array[:-1], new_branch)
93          get_view.data = compact_to_verbose(new)[0]
94
95      socketio.emit('update json', json.dumps(get_view.data, indent=2,
            separators=(', ', ': ')), namespace='/socket')
96
97
98  # ВЫВОД ЛОГОВ
99  print(bot.get_me())
100
101
102 def log(message, answer):
103     """Функция-логгер
104
105     Keyword arguments:
106     message — объект полученного сообщения
```

```
107     answer — текст ответа
108
109     """
110     print("\n --------")
111     from datetime import datetime
112     print(datetime.now())
113     print("Сообщение от {0} {1}. (id = {2}) "
114           "\n Текст - {3}". format(message.from_user.first_name,
115                                    message.from_user.last_name,
116                                    str(message.from_user.id),
117                                    message.text))
118     print(answer)
119
120
121 @socketio.on('connect', namespace='/socket')
122 def handler_connect():
123     """Функция обработчика подключения клиента
124
125     """
126     emit('Connect response', {'data': 'Connected'})
127
128
129 @socketio.on('disconnect', namespace='/socket')
130 def handler_disconnect():
131     """Функция обработчика отключения клиента
132
133     """
134     print('Client disconnected')
135
136
137 @socketio.on('receive answer', namespace='/socket')
138 def get_javascript_data(text):
139     """Функция обработчика полученного ответа с веб-клиента
140
141     Keyword arguments:
142     text — текст сообщения
143
144     """
145     answer_array.append(text)
146
147     r_server = redis.StrictRedis('localhost', charset="utf-8",
            decode_responses=True)
148     qty = r_server.dbsize()
149     for ans in answer_array:
150         r_server.lpush('qa_' + str(qty + 1), ans)
151
152     tg_send(text)
153
154
155 @async()
156 def tg_send(text):
157     """Функция асинхронной отправки сообщения
158
159     Keyword arguments:
160     text — текст сообщения
161
162     """
163     bot.send_message(current_chat_id, text)
```

```
164 |
165 |
166 | @app.route('/viewer/')
167 | def get_view():
168 |     """Функция рендеринга начального экрана
169 |
170 |     """
171 |     get_view.data = {
172 |         "title": "Ожидание диалога"
173 |     }
174 |     return render_template('index.html',
175 |                            context=json.dumps(get_view.data, indent
    |                                =2, separators=(', ', ': ')))
176 |
177 |
178 | @bot.message_handler(commands=['start'])
179 | def handle_start(message):
180 |     """Функция обработчика запуска бота
181 |
182 |     Keyword arguments:
183 |     message — объект сообщения
184 |
185 |     """
186 |     answer = "Начало диалога"
187 |     log(message, answer)
188 |     keyboard = types.ReplyKeyboardMarkup(True, False)
189 |     keyboard.add(*[types.KeyboardButton('Начать диалог')])
190 |     bot.send_message(message.chat.id,
191 |                      """Здравствуйте, вас приветствует система консу
    |                          льтации.""",
192 |                      reply_markup=keyboard,
193 |                      parse_mode="Markdown")
194 |
195 |
196 | @bot.message_handler(func=lambda message: message.text == 'Начать ди
    |  алог', content_types=['text'])
197 | def handle_begin(message):
198 |     """Функция обработчика сообщения "Начать диалог"
199 |
200 |     Keyword arguments:
201 |     message — объект сообщения
202 |
203 |     """
204 |     keyboard = types.ReplyKeyboardRemove()
205 |     answer = """Что вас интересует?"""
206 |     bot.send_message(message.chat.id,
207 |                      answer,
208 |                      reply_markup=keyboard,
209 |                      parse_mode="Markdown")
210 |
211 |     log(message, answer)
212 |
213 |
214 | @bot.message_handler(func=lambda message: message.text != '',
    |  content_types=['text'])
215 | def handle_text(message):
216 |     """Функция обработчика сообщения
217 |
```

```
    Keyword arguments:
    message — объект сообщения

    """
    answer_array.append(message.text)
    answer = """Пожалуйста, подождите."""
    bot.send_message(message.chat.id, answer)

    log(message, answer)

    update_json(answer_array, message.chat.id)


def bot_thread():
    """Функция для запуска треда бота

    """
    bot.polling(none_stop=True)


if __name__ == "__main__":
    _thread.start_new_thread(bot_thread, ())
    socketio.run(app)
```

Листинг Г.2. ml.py

```
from pytils import translit
from pymorphy2 import tokenizers, MorphAnalyzer
import re
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from heapq import nlargest
import numpy as np
import redis

# Регулярное выражение для латинских букв
latin_pattern = r'[A-Za-z]+'
# MorphAnalyzer для нормализации слов
morph = MorphAnalyzer()


# Класс для хранения соответствия вопросных сообщений и ответных
class Question(object):
    """Объект для хранения сообщения пользователя и ответа или поясн
        ящего вопроса для него

    """
    def __init__(self, text, answer):
        self.text = text
        self.answer = answer


def str_handler(in_string):
    """Обработчик строк. Удаление лишних символов, детранслитерация,
        нормализация(приведение к инфинитиву).

    Keyword arguments:
```

```
30      in_string — строка для обработки
31
32      """
33      tokens = tokenizers.simple_word_tokenize(re.sub('[!?,.%]', '',
            in_string))
34      new_string = ''
35      for word_in in tokens:
36          if re.search(latin_pattern, word_in):
37              word_in = translit.detranslify(word_in)
38          new_string += morph.parse(word_in)[0].normal_form + ' '
39      return new_string
40
41
42  def get_most_probable(profiler, v):
43      """Функция возвращает 3 наиболее вероятных ответа.
44
45      Keyword arguments:
46      profiler — объект обученного случайного леса
47      v — вектор векторизированного вопроса
48
49      """
50
51      # Предсказание вероятностей каждого ответа для каждого вопроса из
            тестовой выборки
52      predict_data = profiler.predict_proba(v)
53      ans = []
54      # Вывод 3 наиболее вероятных ответов
55      for i in predict_data:
56          for j in i:
57              if j in nlargest(3, i):
58                  ans.append(profiler.classes_[i.tolist().index(j)])
59      return ans
60
61
62  def preprocessing(answer_array):
63      """Функция, которая подготавливает выборку в зависимости текущег
            о состояния массива диалога
64
65      Keyword arguments:
66      answer_array — Вектор сообщений в текущем диалоге
67
68      """
69
70      r_server = redis.StrictRedis('localhost', charset="utf-8",
            decode_responses=True)
71
72      qa = []
73      for key in r_server.scan_iter():
74          row = r_server.lrange(key, 0, r_server.llen(key))
75          row.reverse()
76
77          if len(answer_array) == 1:
78              if len(row) == 2:
79                  qa.append(Question(row[-2], row[-1]))
80          else:
81              if answer_array[0:-1] == row[0:-2]:
82                  qa.append(Question(row[-2], row[-1]))
83
```

```
84      if not qa:
85          return []
86      else:
87          return analyze(qa, answer_array[-1])
88
89
90  def analyze(questions, args):
91      """Функция анализа базы вопросов и последующего предсказания
92
93      Keyword arguments:
94      questions — вектор объектов вопрос—ответ
95      args — заданный вопрос
96
97      """
98
99      # Создание TFID векторайзера
100     vectorizer = TfidfVectorizer(min_df=1)
101
102     # Обработка строк вопросов
103     corpus = []
104     for qst in questions:
105         corpus.append(str_handler(qst.text))
106
107     # Обучение векторайзера на уже подготовленной базе
108     vectorizer.fit_transform(corpus)
109
110     data_target_tuples = []
111     for qst in questions:
112         data_target_tuples.append((str_handler(qst.text), qst.answer
                ))
113
114     # Трансформация вопросов в матричный вид, на основе TFID критерия
115     x_data = []
116     y_target = []
117     for t in data_target_tuples:
118         v = (vectorizer.transform([t[0]]).toarray())[0]
119         x_data.append(v)
120         y_target.append(t[1])
121
122     x_data = np.asarray(x_data)
123     y_target = np.asarray(y_target)
124
125     # Обучение случайного леса с заданными параметрами
126     profiler = RandomForestClassifier(max_depth=15, n_estimators=36,
            max_features=2)
127     profiler.fit(x_data, y_target)
128
129     v = (vectorizer.transform([str_handler(args)]).toarray())[0]
130
131     return get_most_probable(profiler, v)
```