

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)

Институт Радиоэлектроники и информационных технологий

Направление подготовки (специальность) 09.03.01 Информатика и вычислительная техника
(код и наименование)

Направленность (профиль) образовательной программы Вычислительные машины, комплексы, системы и сети
(наименование)

Кафедра Вычислительных систем и технологий

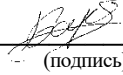
ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
бакалавра

(бакалавра, магистра, специалиста)

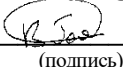
Студента Барина Роман Олеговича группы 16-B-2
(Ф.И.О.)

на тему Программная система с естественно-языковым интерфейсом
(наименование темы работы)

СТУДЕНТ:

 Барин Р. О.
(подпись) (фамилия, и., о.)
03.07.2020
(дата)

РУКОВОДИТЕЛЬ:

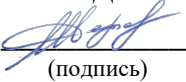
 Гай В. Е.
(подпись) (фамилия, и., о.)
03.07.2020
(дата)

РЕЦЕНЗЕНТ:

(подпись) (фамилия, и., о.)

(дата)

ЗАВЕДУЮЩИЙ КАФЕДРОЙ

 Жевнерчук Д. В.
(подпись) (фамилия, и.о.)
03.07.2020
(дата)

КОНСУЛЬТАНТЫ:

1. По _____

(подпись) (фамилия, и., о.)

(дата)

2. По _____

(подпись) (фамилия, и., о.)

(дата)

3. По _____

(подпись) (фамилия, и., о.)

(дата)

ВКР защищена _____
(дата)

протокол № _____

с оценкой _____

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

Кафедра Вычислительных систем и технологии



УТВЕРЖДАЮ
Зав. кафедрой
Жевнерчук Д. В.

«12» мая 2020 г.

ЗАДАНИЕ

на выполнение выпускной квалификационной работы

по направлению подготовки (специальности) 09.03.01 Информатика и вычислительная техника

студенту Баринову Роману Олеговичу группы 16-В-2
(Ф.И.О.)

1. Тема ВКР Программная система с естественно-языковым интерфейсом (утверждена приказом по вузу от 15.04.2020 № 867/5)

2. Срок сдачи студентом законченной работы 03.07.2020

3. Исходные данные к работе справочная система ВУЗа; язык разработки – Python; система включает программу для операционной системы Linux.

4. Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке)

Введение

1. Техническое задание

2. Анализ технического задания

3. Разработка системы на структурном уровне

4. Разработка программных средств

5. Тестирование системы

Заключение

Список литературы

Приложения

5. Перечень графического материала (с точным указанием обязательных чертежей)

1. Структурная схема разрабатываемой системы

2. Структурные схемы работы программных алгоритмов

3. Структурная схема модели нейронной сети

4. Тестовые изображения

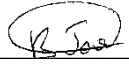
5. Результаты работы программы

6. Консультанты по ВКР (с указанием относящихся к ним разделов)

Нормоконтроль Гай В.Е.

7. Дата выдачи задания 12.05.2020

Код и содержание Компетенции	Задание	Проектируемый результат	Отметка о выполнении
ПК-1, Способность разрабатывать модели компонентов информационных систем, включая модели баз данных и модели интерфейсов «человек – электронновычислительная машина»	Разработать структурную схему системы	Структурная схема системы	
ПК-2, Способность разрабатывать компоненты аппаратно-программных комплексов и баз данных, используя современные инструментальные средства и технологии программирования	Разработать алгоритмы работы системы, реализовать их на одном из языков программирования с помощью современной среды разработки	Схема алгоритмов работы, реализация на языке Python с помощью среды PyCharm	
ПК-3, Способность обосновывать принимаемые проектные решения, осуществлять постановку и выполнять эксперименты по проверке их корректности и эффективности	Осуществить отбор лучших результатов, оценить их качество по нескольким критериям	Осуществлен анализ результатов, оценено их качество	

Руководитель  Гай В.Е.
(подпись)

Задание принял к исполнению 12.05.2020
(дата)

Студент  Баринов Р. О.
(подпись)

**МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМ. Р.Е. АЛЕКСЕЕВА»
(НГТУ)**

АННОТАЦИЯ

к выпускной квалификационной работе

по направлению подготовки (специальности) 09.03.01

Информатика и

вычислительная техника

(код и наименование)

Студента Барина Роман Олегович группы 16-В-2
(Ф.И.О)

по теме Программная система с естественно-языковым интерфейсом.

Выпускная квалификационная работа выполнена на 59 страницах, содержит 14 рисунков, 0 таблиц, библиографический список из 8 источников, 1 приложение.

Актуальность: необходимость создания программной системы с естественно-языковым интерфейсом для помощи абитуриентам и студентам младших курсов НГТУ им. Р. Е. Алексеева в связи с ростом популярности подобных систем и отсутствием аналогов в данной сфере.

Объект исследования: текстовая информация из справочных источников ВУЗа.

Предмет исследования: алгоритмы и методы классификации текстовых запросов к справочной системе.

Цель исследования: разработка программной системы с естественно-языковым интерфейсом.

Задачи исследования: исследовать существующие методы построения систем с естественно-языковым интерфейсом; исследовать методы классификации текстовых запросов к справочной системе; разработать собственную справочную систему для помощи абитуриентам и студентам младших курсов НГТУ им. Р. Е. Алексеева; разработать собственную программную систему для классификации текстовых запросов к справочной системе и поиска конечного ответа; выполнить тестирование с целью проверки работоспособности разработанной программной системы.

Методы исследования: классификация текстовых запросов на основе модели нейронной сети и метода «bag of words»; Поиск конечного ответа в справочной системе на основе метода tf-idf и косинусного коэффициента.

Структура работы: выпускная квалификационная работа состоит из введения, пяти глав, заключения, списка литературы и приложения.

Во введении дается описание проблемы, лежащей в основе данной работы.

В 1 разделе «Техническое задание» составлено техническое задание на разработку.

Во 2 разделе «Анализ технического задания» производится выбор программных средств, операционной системы, обзор существующих методов построения систем с естественно-языковым интерфейсом, обзор существующих методов классификации текстовых запросов, приводится обоснование выбора метода построения системы с естественно-языковым интерфейсом и метода классификации текстовых запросов.

В 3 разделе «Разработка системы на структурном уровне» разрабатывается структурная схема, алгоритмы решения каждого из этапов решения задачи.

В 4 разделе «Разработка программных средств» разрабатываются программные средства для решения поставленной задачи.

В 5 разделе «Тестирование системы» описывается метод тестирования системы и полученные результаты.

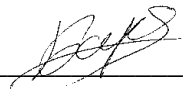
В заключении приводятся основные выводы по работе.

Выводы:

1. Разработана программная система с естественно-языковым интерфейсом
2. Тестирование системы подтвердило её работоспособность и возможность использования для решения поставленной задачи.





Рекомендации:

1. Дальнейшее развитие проекта.
2. Оптимизация существующего кода.

 / Баринов Р. О. подпись студента /расшифровка подписи
«02» июля 2020 г.

Оглавление

Введение	5
1. Техническое задание	6
1.1 Назначение разработки и область применения.....	6
1.2 Технические требования	6
2. Анализ технического задания	8
2.1 Выбор операционной системы	8
2.2 Выбор языка программирования.....	9
2.3 Обзор существующих методов построения систем с естественно-языковым интерфейсом.....	12
2.4 Обзор существующих методов классификации пользовательских запросов	12
2.5 Выбор метода построения системы с естественно-языковым интерфейсом и метода классификации запроса пользователя.....	14
3. Разработка системы на структурном уровне	15
3.1 Разработка структурной схемы системы	15
3.2 Разработка алгоритма предварительной обработки запроса	16
3.3 Разработка алгоритма вычисления признакового описания	18
3.4 Разработка алгоритма классификации запроса.....	19
3.5 Разработка алгоритма поиска ответа на запрос	21
3.6 Разработка словаря ответов	24
4. Разработка программных средств	26
4.1 Модуль предварительной обработки	26
4.2 Модуль вычисления признакового описания	28
4.3 Модуль классификации запроса.....	30
4.4 Модуль поиска ответа на запрос	34
4.5 Модуль взаимодействия с ПО «Telegram Messenger».....	37
5. Тестирование системы	39
Заключение.....	42
Список литературы.....	43
Приложение.....	44

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)			
Изм.	Лист	№ докум.	Подпись	Дата	Программная система с естественно-языковым интерфейсом Пояснительная записка	Лит.	Лист	Листов
Разраб.		Баринков Р.О.		03.07			4	59
Провер.		Гай В.Е.		03.07				
Т. контр.								
Н. контр.		Гай В.Е.		03.07				
Утв.		Жевнерчук Д.В.		03.07		НГТУ кафедра ВСТ		

Введение

В настоящее время существует большое количество различных систем, помогающих пользователям ориентироваться в городе, находить необходимые места и транспорт. Однако, данные системы направлены, в первую очередь, на решение проблем геопозиционирования и поиска необходимого транспорта и практически не предоставляют пользователям конкретной справочной информации, за исключением телефонных номеров организаций, времени работы и ссылок на веб-сайты. В свою очередь, на веб-сайтах организаций, в большинстве случаев, есть необходимая справочная информация, однако, пользователь вынужден тратить время на поиск нужных ему данных.

Абитуриенты и первокурсники высших учебных заведений, в частности, НГТУ им. Р. Е. Алексеева, часто сталкиваются с проблемами, связанными с ориентацией в корпусах, поиском расписания учебных занятий, медкабинетов или банкоматов, а также необходимых им справочных телефонов кафедр, деканатов, студенческих объединений или комендантов общежитий и т.п.

Естественно, веб-сайт нашего ВУЗа, как и многих других, позволяет найти большую часть справочной информации, но далеко не всю, и зачастую для её поиска приходится тратить немалое время, переходя по многочисленным вкладкам и страницам сайта.

Задача, на решение которой направлена данная работа – разработка программной системы с естественно-языковым интерфейсом для помощи абитуриентам и первокурсникам НГТУ им. Р. Е. Алексеева, включающую в себя всю необходимую справочную информацию, связанную с университетом, студгородком и общежитиями, транспортом и маршрутах проезда в одном месте.

1. Техническое задание

1.1 Назначение разработки и область применения

Разрабатываемая программная система предназначена для выполнения поиска и предоставления информации, по запросу пользователя на естественном языке, в сформированной базе данных, связанной с НГТУ им. Р.Е. Алексеева.

Область применения разрабатываемой системы:

- Помощь абитуриентам и студентам младших курсов в поиске необходимой справочной информации, связанной с ВУЗом.

1.2 Технические требования

Разрабатываемая система с естественно-языковым интерфейсом должна классифицировать пользовательские запросы и на основе класса запроса выполнять поиск необходимой информации в соответствующем классе.

Данная система предполагает наличие клиент-серверного взаимодействия. Клиент-серверное взаимодействие должно быть реализовано на основе ПО «Telegram Messenger». Причем, предполагается наличие двух типов серверов.

Первый тип – это сервера, обеспечивающие работу сетевой структуры системы, то есть приём и пересылку (если это необходимо) сообщений с клиентских машин на сервер второго типа (непосредственного обработчика запроса). Реализация данных серверов не входит в настоящее ТЗ, данные сервера – это сервера компании «Telegram Messenger», именно на их основе будет реализована сетевая структура системы.

Второй тип – это сервер, обеспечивающий непосредственную обработку естественного языка. На стороне данного сервера будут происходить основные вычислительные действия по обработке пользовательского запроса. Данный сервер должен быть реализован с учётом согласования с ПО «Telegram Messenger». Пользовательские запросы на сервер второго типа будут поступать от сервера первого типа. Конечные ответы пользователю будут возвращаться на сервер первого типа для дальнейшей пересылки на клиентскую часть.

Клиентская часть системы должна обеспечивать возможность ввода пользовательского запроса, отправку запроса на сервер первого типа, приём результата от сервера первого типа и вывод его пользователю. Клиентская часть системы должна быть реализована с помощью ПО «Telegram Messenger».

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						6
Изм	Лист	№ докум.	Подп.	Дата		

Клиентом и сервером второго типа в данном взаимодействии должны выступать ЭВМ, поддерживающие ОС, на которых есть возможность установки ПО «Telegram Messenger».

Рассмотрим функциональность, которой должна обладать данная система:

- 1) Пользователь системы должен иметь возможность ввода запроса на естественном языке;
- 2) Векторизация пользовательского запроса;
- 3) Классификация пользовательского запроса;
- 4) Поиск ответа на запрос в базе данных внутри определённого класса;
- 5) Предоставление ответа пользователю на естественном языке.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						7
Изм	Лист	№ докум.	Подп.	Дата		

2. Анализ технического задания

2.1 Выбор операционной системы

Так как разрабатываемая система подразумевает наличие клиентской и серверной ЭВМ, необходимо определить какая операционная система будет установлена на сервере, а какие ОС подойдут для установки на клиентах данной системы.

Для клиентских ОС нет особых требований, за исключением требования по поддержке ПО «Telegram Messenger». Следовательно, клиентской операционной системой может быть любая ОС, удовлетворяющая данному требованию.

Рассмотрим варианты семейств операционных систем, которые подходят для реализации серверной части разрабатываемой программной системы:

1) Linux – это семейство Unix-подобных операционных систем с открытым исходным кодом, основанных на ядре Linux. Распространяется в основном бесплатно в соответствии с моделью разработки свободного и открытого программного обеспечения. У Linux нет единой комплектации, так как она поставляется в виде дистрибутивов. В состав дистрибутива входят ядро Linux, вспомогательное системное программное обеспечение и библиотеки, многие из которых предоставляются проектом GNU.

2) Windows – представляет собой группу из нескольких проприетарных семейств операционных систем, все из которых разрабатываются и продаются компанией Microsoft. Активные семейства Microsoft Windows включают Windows NT и Windows IoT. Различные версии ОС Windows получили большую популярность среди пользователей. Это можно объяснить ориентированностью на управление с помощью графического интерфейса. Главным недостатком современных версий ОС Windows является их проприетарность и как следствие цена.

3) MacOS – это серия проприетарных операционных систем, разработанная и продаваемая компанией Apple Inc. Это основная операционная система для компьютеров Apple Mac. MacOS - вторая основная серия операционных систем Macintosh. MacOS основана на операционной системе Unix и на технологиях, разработанных в 1985–1997 годах в NeXT. Сертификация UNIX 03 была достигнута для MacOS X 10.5 Leopard и все выпуски от Mac OS X 10.6 Snow Leopard до текущей версии также имеют сертификацию UNIX 03.

Исходя из поставленных требований и обзора данных семейств операционных систем можно сделать вывод, что наиболее подходящей системой для реализации серверной части разрабатываемой системы является ОС Linux, в частности Ubuntu –

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						8
Изм	Лист	№ докум.	Подп.	Дата		

операционная система, основанная на Debian GNU/Linux. Основной причиной выбора данной ОС является её бесплатное распространение и личный опыт использования.

2.2 Выбор языка программирования

Важным этапом в процессе разработки программной системы является этап выбора языка программирования. Данный выбор зависит от требований, предъявляемых к разрабатываемой системе. В первую очередь язык программирования должен удовлетворять требованию по поддержке инструментов машинного обучения для возможности использования средств классификации пользовательских запросов.

Рассмотрим наиболее подходящие под данное условие языки программирования:

1) Python – высокоуровневый язык программирования, обладающий динамической типизацией и автоматическим управлением памятью. Python поддерживает объектно-ориентированное, императивное, структурное, функциональное и аспектно-ориентированное программирование. Важнейшим преимуществом языка Python является его обширная стандартная библиотека, включающая в себя большой объём полезных функций. Python также поддерживает такие библиотеки машинного обучения как: TensorFlow, Keras, scikit-learn и др. Причем, именно для языка Python разработан основной API для работы с библиотекой машинного обучения TensorFlow.

2) C++ – высокоуровневый язык программирования, компилируемый, статически типизированный язык программирования, поддерживающий основные парадигмы программирования, такие как объектно-ориентированное программирование, процедурное программирование, обобщённое программирование, а также поддерживает средства низкоуровневой манипуляции с памятью. C++ был разработан с уклоном в сторону системного программирования, встроенного программного обеспечения с ограниченными ресурсами и больших систем. Язык C++ зачастую применяется и во многих других аспектах, где основным требованием является разработка приложений в условиях с ограниченными ресурсами.

3) Haskell – стандартизированный, статически типизированный, функциональный язык программирования общего назначения. Является одним из самых распространённых языков программирования с поддержкой отложенных вычислений. Классы типов, например, включающие безопасную перегрузку операторов, были впервые предложены Филиппом Уодлером и Стивеном Блоттом для Standard ML и впервые реализованы в Haskell.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						9
Изм	Лист	№ докум.	Подп.	Дата		

Имеются средства взаимодействия с кодом на других языках программирования. Есть встроенная поддержка многозадачного и параллельного программирования, развитый инструментарий (средства автоматического тестирования, отладки и профилирования).

4) Java – строго типизированный объектно-ориентированный язык программирования. Язык Java спроектирован таким образом, чтобы иметь как можно меньше зависимостей реализации. Скомпилированный код Java может работать на всех платформах, поддерживающих Java, без необходимости перекомпиляции. Приложения Java компилируются в байт-код, который может работать на любой виртуальной машине Java (JVM) независимо от базовой архитектуры компьютера. Синтаксис Java похож на C и C++, однако у языка Java меньше низкоуровневых возможностей.

Рассмотрев преимущества и недостатки вышеописанных языков программирования, а также учитывая требования по поддержке инструментов машинного обучения, остановимся с выбором на высокоуровневом языке программирования Python, который удовлетворяет заданным требованиям, имеет гибкую и многофункциональную стандартную библиотеку.

Рассмотрим библиотеки выбранного языка программирования Python, которые необходимы нам для реализации программной системы с естественно-языковым интерфейсом:

1) NumPy – это фундаментальный пакет для научных вычислений на языке Python с открытым исходным кодом. Обеспечивает поддержку работы с матрицами и линейной алгеброй, а также является альтернативой MatLab.

2) Pandas – данная библиотека написана на языке программирования Python для манипулирования и анализа данных. Обладает полезными инструментами для обработки данных и позволяет выполнять:

- манипулирование данными со встроенной индексацией;
- чтение и запись данных между структурами данных в памяти и различными форматами файлов;
- выравнивание данных;
- обработку отсутствующих данных;
- изменение формы и поворот наборов данных;
- срезы на основе меток.

3) Pickle – модуль, реализующий двоичные протоколы для сериализации и десериализации структуры объекта Python.

4) NLTK [1] – это пакет библиотек для символьной и статистической обработки естественного языка. NLTK предоставляет простые в использовании интерфейсы для лексических ресурсов, таких как WordNet, наряду с набором библиотек обработки текста для классификации, токенизации, обработки по меткам, разметки, анализа и семантического мышления.

5) Keras – это открытая, бесплатная нейросетевая библиотека Python с открытым исходным кодом для разработки и оценки моделей глубокого обучения. Данная библиотека является надстройкой над библиотеками числовых вычислений Theano и TensorFlow и позволяет определять и обучать модели нейронных сетей.

6) Matplotlib – библиотека для визуализации данных. Поддерживается двухмерная и трёхмерная графика. Данная библиотека распространяется на условиях BSD-подобной лицензии.

7) Sklearn (scikit-learn) – бесплатная библиотека машинного обучения для языка программирования Python. Sklearn имеет различные алгоритмы классификации, регрессии и кластеризации, включая метод опорных векторов, случайный лес, повышение градиента, k-ближайших соседей, DBSCAN и др., и предназначена для взаимодействия с числовыми и научными библиотеками Python такими как NumPy и SciPy.

8) Netron – это библиотека, выступающая в качестве средства просмотра моделей нейронных сетей, глубокого обучения и машинного обучения.

9) Telebot – библиотека, позволяющая создавать так называемых Telegram-ботов для одноименного мессенджера, включающая в себя декораторы маршрутов.

2.3 Обзор существующих методов построения систем с естественно-языковым интерфейсом

Существует две основные модели построения систем с естественно-языковым интерфейсом [3]: генеративная модель и выборочная (поисковая) модель.

1) Генеративная модель:

- может генерировать произвольный ответ;
- склонна к несоответствию ответа и поставленного вопроса;
- может генерировать ответ в ошибочной грамматической и синтаксической форме;
- требуется большое число данных для обучения модели;

2) Выборочная (поисковая) модель:

- ограниченный набор заготовленных ответов;
- всегда генерирует ответ в правильной (заданной) грамматической и синтаксической форме;
- не требуется большое число данных для обучения модели;

2.4 Обзор существующих методов классификации пользовательских запросов

Основной проблемой, с которой придётся столкнуться при разработке программной системы взаимодействия с пользователем на естественном языке, является проблема классификации намерения (смысла или интента) пользователя, то есть определение смысла запроса. Успех дальнейшего поиска или генерации ответа на пользовательский запрос напрямую зависит от правильности определения интента запроса.

Рассмотрим существующие методы классификации интента пользовательского запроса:

1) Классификация на основе словаря [3, 5]. Данный метод является самым простым методом классификации пользовательских запросов и подразумевает наличие словаря интентов. Суть классификации на основе словаря заключается в поиске в пользовательском запросе слов из словаря интентов. Достоинством данного метода является отсутствие необходимости создания размеченной выборки. Однако этот метод никак не учитывает перефразирование запросов, а также возникают трудности с классификацией синонимичных и очень похожих по написанию, но различных по смыслу запросов.

2) Классификация на основе метода Bag of Words [7]. Пользовательский запрос проходит предварительную обработку – векторизацию, то есть преобразование словесного

запроса в числовой вектор, на основе вхождения слов запроса в общий словарь ответов. Метод основан на принципе схожести текстов. Далее, полученный вектор поступает на вход предварительно обученной нейронной сети, которая возвращает вероятность принадлежности входного вектора, а значит и пользовательского запроса, к определённому классу (интенту). В данном случае нейронная сеть проходит обучение на векторизованных тем же способом предложениях из словаря ответов.

Преимуществом данного метода является относительная простота его реализации при большом проценте точности на небольшом количестве классов.

3) Метод основанный на модели ELMo. Суть модели ELMo заключается в построении для каждого слова в запросе посимвольного эмбединга слова (то есть, преобразования слова в числовой вектор) и дальнейшем применении LSTM-сети [8]. Таким образом, получается модель, учитывающая контекст, в котором встретилось слово.

Посимвольный эмбединг получается путём разбивания слова на символы и дальнейшего применения для каждого символа эмбединг-слоя, в итоге получается матрица эмбедингов. Далее, к матрице эмбедингов применяются одномерная свертка и получается один вектор. К этому вектору применяется двухслойная, так называемая, highway-сеть, которая вычисляет общий вектор слова. В таком случае, модель построит гипотезу эмбединга, другими словами векторизует слово, причём даже такое, которое не встречалось в обучающей выборке.

После вычисления посимвольных эмбедингов для каждого слова, необходимо применить к ним двухслойную BiLSTM-сеть. После применения двухслойной BiLSTM-сети берутся hidden states последнего слоя, то есть контекстные эмбединги.

Безусловным преимуществом данной модели является контекстная зависимость векторов слов. Однако, чрезмерная сложность построения и обучения модели делает её оправданно применимой только в задачах с большим количеством интентов. Ещё один недостаток такого подхода заключается в том, что данная модель не даёт гарантий, что тексты, которые относятся к одному классу, то есть, к одному интенту, будут близки в векторном пространстве.

2.5 Выбор метода построения системы с естественно-языковым интерфейсом и метода классификации запроса пользователя

Разрабатываемая программная система, согласно области применения, должна иметь небольшое число пользовательских интенгов, то есть тем (классов), связанных только с основными проблемами абитуриентов и студентов младших курсов в области получения справочной информации о структурах ВУЗа, студенческого городка и транспорта.

В условиях ограниченного числа классов использование генеративной модели построения системы с естественно-языковым интерфейсом явно нецелесообразно, так как влечёт за собой необходимость в большом количестве данных для обучения модели, а также, имитируя диалог, склонна к несоответствию ответа и поставленного вопроса, грамматических и синтаксических форм, что делает её не комфортной для конечного пользователя, который намерен получить конкретную информацию по своему запросу.

Таким образом, при построении данной системы будет использоваться выборочная модель, из-за её преимущества в генерации ответа в правильной грамматической и синтаксической форме (данное преимущество обусловлено наличием заготовленной базы данных на соответствующую поставленной задаче тематику), а также из-за отсутствия необходимости большого числа данных для обучения такой модели.

Что касается выбора алгоритма классификации пользовательского запроса, то исходя из анализа описанных выше методов, целесообразно остановится на классификации с применением нейронной сети и методом векторизации – Bag of Words [7]. Данный способ классификации не требует построения громоздкого математического аппарата, как метод на основе модели ELMo, включающий в себя две нейронные сети.

3. Разработка системы на структурном уровне

3.1 Разработка структурной схемы системы

Систему с естественно-языковым интерфейсом для помощь абитуриентам и студентам младших курсов НГТУ им. Р. Е. Алексеева можно представить следующим образом (рис. 1)



Рисунок 1. Структурная схема разрабатываемой системы.

На вход системе подаётся очередной пользовательский запрос на естественном языке. Алгоритм проводит предварительную обработку принятого запроса для подготовки данных для векторизации, то есть вычисления признакового описания. После вычисления признакового описания происходит классификация пользовательского запроса, с помощью обученной заранее модели. На основе определённого класса алгоритм производит поиск ответа на заданный запрос в базе данных по этому классу.

3.2 Разработка алгоритма предварительной обработки запроса

Предварительная обработка принятого запроса должна выполняться на серверной ЭВМ. Такая обработка данных на естественном языке состоит из следующих этапов:

1) Токенизация запроса. Процесс токенизации или как его еще называют процесс сегментации – это процесс разделения однородной структуры (текста или отдельных предложений), написанной на естественном языке, на более мелкие структуры – предложения или слова.

Процесс разделения текста на предложения на первый взгляд кажется тривиальным. Нашли точку, знак вопроса и тд. – определили конец предложения, однако это не всегда верное решение. Ведь точки употребляются и при сокращениях, например, вопрос «кто является ректором НГТУ им. Р. Е. Алексеева?» – это одно предложение, а не четыре отдельных. В таком случае, чтобы избежать неверного разделения текста на предложения необходимо учитывать данную особенность построения предложений. В большинстве случаев для этого используются специальные библиотеки, в которых заложены функции определения сокращений.

Другой процесс токенизации – это процесс разделения предложений на слова. В этом случае надёжным признаком конца слова является символ пробел, естественно, если имеется в виду русскоязычная лексика и грамматика.

В разрабатываемой системе нет необходимости использовать оба метода токенизации, так как запросы о поиске справочной информации, в определённой заранее группе тем, строятся из одного предложения, а значит необходимо использовать только второй процесс токенизации – токенизация предложения на слова.

2) Лемматизация и стемминг [4, 5] запроса. Обычно в предложениях на естественном языке встречаются различные формы одного и того же слова, а также однокоренные слова. Оба процесса – лемматизация и стемминг, необходимы для приведения слов к их словарной (нормальной) форме. Однако, между данными процессами есть разница.

Лемматизация – это процесс, который использует словарь и морфологический анализ, чтобы привести слово к лемме, то есть к словарной форме. В свою очередь, стемминг – это эвристический процесс, который убирает некоторые словообразующие морфемы от корня слова. Такой алгоритм является более грубым в сравнении с лемматизацией и действует без знания контекста. Соответственно, такой алгоритм не понимает разницу между словами, которые имеют разный смысл в зависимости от части речи. Однако алгоритм стемминга обладает явным преимуществом для решения нашей

задачи, так как является более быстродействующим в сравнении с лемматизацией из-за отсутствия в библиотеке алгоритма крупных словарей.

3) Удаление «стоп» слов. «Стоп» слова – это слова, которые удаляются из обрабатываемого текста. В случаях, когда необходимо применить алгоритмы машинного обучения к текстам или отдельным предложениям, некоторые слова могут добавить много шума, то есть при дальнейшей векторизации иметь слишком большие веса и мешать обработке запросов. Под такими словами обычно понимают артикли, междометия, союзы и т.д., то есть слова, которые не несут смысловой нагрузки. Однако при применении алгоритмов удаления «стоп» слов необходимо понимать, что не существует универсального списка «стоп» слов. Все зависит от конкретного языка, области применения и тематики.

Таким образом, предварительная обработка пользовательского запроса состоит из трёх алгоритмов обработки естественного языка – токенизации, стемминга и удаления «стоп» слов. Соответственно, на вход алгоритму предварительной обработки поступает непосредственно предложение, написанное пользователем, а возвращается список стемм слов.

Так как, в дальнейшем стоит необходимость в применении средств машинного обучения для вычисления вероятности принадлежности запроса к определённому классу, то все три описанных выше алгоритма предварительной обработки необходимо применить и к базе данных ответов на запросы, для обучения модели нейронной сети. Эта операция, в отличие от предварительной обработки пользовательских запросов, выполняется однократно и повторяется только в случае необходимости переобучения модели, то есть при расширении информационной системы (добавлении новых классов или изменении текущих). Обобщённый алгоритм предварительной обработки представлен на рисунке 2.



Рисунок 2. Алгоритм предварительной обработки пользовательского запроса.

3.3 Разработка алгоритма вычисления признакового описания

Признаковое описание запроса, как и предварительная обработка, выполняется на серверной ЭВМ. Под признаковым описанием пользовательского запроса следует понимать его преобразование в вид, который необходим для дальнейшей обработки запроса инструментами машинного обучения. Другими словами, признаковое описание – это векторизация запроса.

Метод векторизации напрямую зависит от используемых инструментов машинного обучения. На этапе выбора метода классификации запроса был выбран метод, который подразумевает применение нейронной сети с векторизацией методом Bag of Words. Данный метод векторизации относительно прост в своей реализации, однако требует наличия заготовленного словаря уникальных слов (в нашем случае им является база данных ответов на запросы, прошедшая предварительную обработку при обучении модели), а так как такая база данных в любом случае необходима нам для формирования ответа пользователю, данное замечание не является недостатком.

Суть метода Bag of Words заключается в формировании вектора, соответствующего входному предложению. В соответствии с каждым словарным словом в векторе ставится ноль или единица, в зависимости от условия вхождения словарных слов в обрабатываемое предложение. При таком подходе величина такого вектора будет равна длине словаря, по которому происходит вычисление вектора. Само вычисление вектора представлено в виде схемы на рисунке 3.

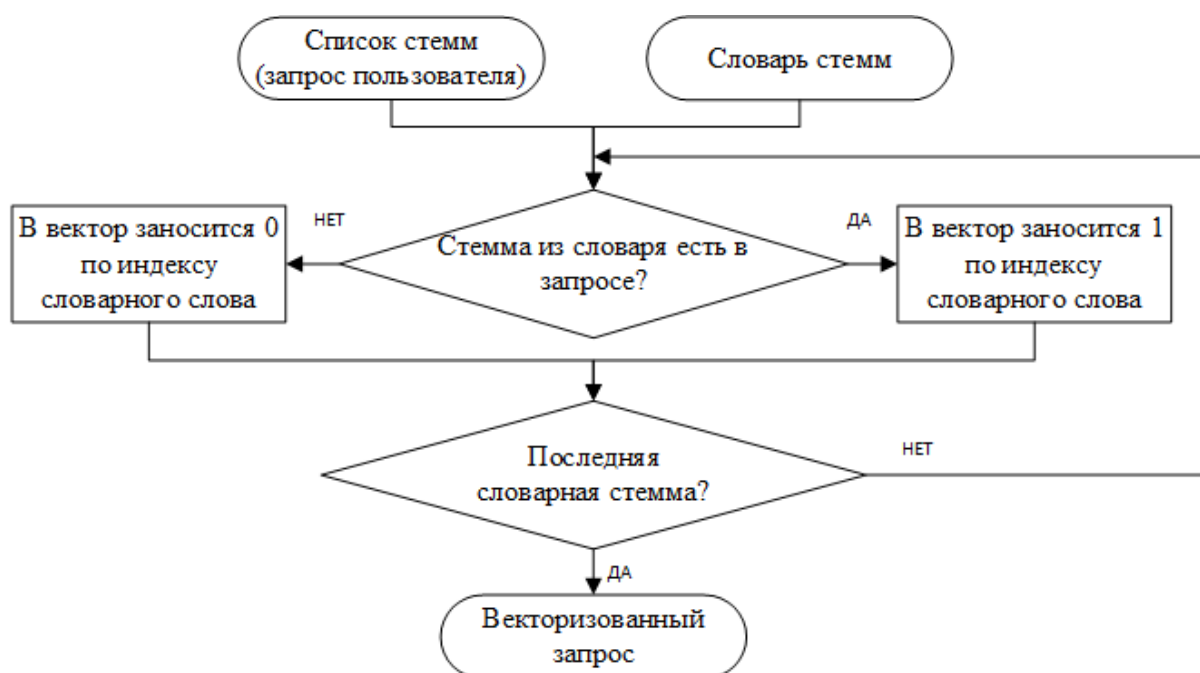


Рисунок 3. Алгоритм векторизации пользовательского запроса.

3.4 Разработка алгоритма классификации запроса

Классификация пользовательского запроса, а именно определение интента, будет выполняться с помощью инструментов машинного обучения. В разрабатываемой системе таким инструментом выступит искусственная нейронная сеть.

Одним из наиболее трудоёмких процессов в создании модели нейронной сети, является процесс обучения и конфигурирования первичных параметров сети, таких как: количество слоёв и количество нейронов на каждом слое.

Процесс обучения нейронной сети – это адаптация модели сети к решаемой задаче. Обучение включает в себя настройку весов сети для повышения точности результата. Это делается путем минимизации ошибок.

В данном случае, при обучении нейронной сети будет применяться метод обучения с учителем. Данный метод подразумевает такой подход, когда для каждого обучающего объекта принудительно задаётся правильный результат (класс). Конфигурирование весов при обучении модели будет выполняться методом градиентного спуска (или как он ещё известен – метод обратного распространения ошибки) с использованием ускоренного градиента Нестерова.

Метод градиентного спуска – это метод нахождения минимального значения функции потерь. Функция потерь используется для контроля ошибки в прогнозах нейронной сети. Поиск минимума функции потерь необходим для получения наименьшей возможной ошибки и повышения точности модели. Суть алгоритма заключается в нахождении наименьшего значения ошибки. Чтобы найти минимальную ошибку в функции потерь, необходимо изменять параметры (веса) модели.

Для минимизации вероятности переобучения вводится так называемый процесс «Dropout». Данный процесс предназначен для предотвращения коадаптаций отдельных нейронов на тренировочных данных во время обучения.

Опытным путём, с учётом минимизации ошибки и увеличения конечной точности работы модели, были определены следующие параметры нейронной сети:

1. Количество слоёв сети – 3;
2. Количество нейронов на каждом уровне сети – 128, 64 и 15 соответственно;

На рисунке 4 представлена структурная схема искусственной нейронной сети.

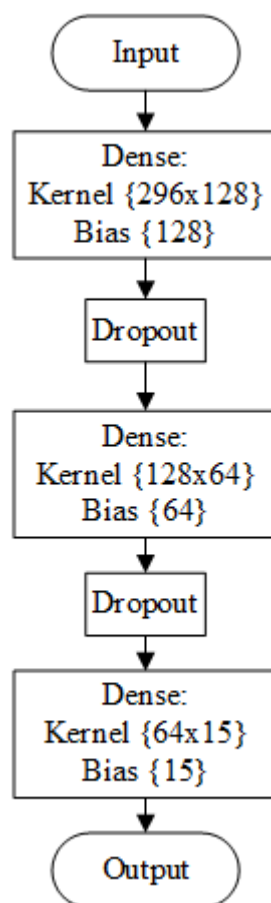


Рисунок 4. Структурная схема нейронной сети.

На вход нейронной сети подаётся вектор признакового описания, полученный в ходе векторизации пользовательского запроса. Далее, данные проходят через 3 слоя сети с разным количеством нейронов. Выходом сети является вектор из 15 элементов, каждый элемент данного вектора является вероятностью принадлежности входного векторизованного пользовательского запроса к определенному классу. Класс входного запроса определяется по наибольшей вероятности, то есть по максимальному элементу выходного вектора.

3.5 Разработка алгоритма поиска ответа на запрос

После процесса классификации запроса и, соответственно, получения класса обработанного запроса необходимо выполнить поиск ответа на заданный пользовательский запрос. Поиск ответа будет осуществляться в базе данных ответов по определённом на предыдущем этапе классу.

Алгоритм поиска ответа на запрос состоит из четырёх этапов:

- 1) Векторизация пользовательского запроса с учётом весов слов;
- 2) Вычисление сходства между векторами (коэффициента Отиаи [6]);
- 3) Выбор наиболее подходящего вектора из БД ответов;
- 4) Интерпретация вектора в ответ на естественном языке.

Рассмотрим каждый из этапов подробнее:

1) Векторизация пользовательского запроса с учётом весов слов. Для того, чтобы минимизировать влияние в запросе вводных слов, междометий, предлогов и союзов, и наоборот, повысить влияние остальных слов, которые нужны для определения ответа необходимо учитывать так называемые веса слов.

Таким методом векторизации является метод TF-IDF [2, 3]. TF-IDF (3) – это статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью набора документов. Данная мера состоит из двух частей.

TF (1) или частота слова – это отношение количества вхождения конкретного термина к суммарному набору слов в исследуемом документе. Этот показатель отражает важность (весомость) слова в рамках определенного документа (записи).

IDF (2) или обратная (инвертированная) частота документа – это инверсия частотности, с которой определенное слово фигурирует в коллекции текстов. Благодаря данному показателю можно снизить весомость наиболее широко используемых слов (предлогов, союзов, вводных слов и т.п.). Для каждого термина в рамках определенной базы текстов предусматривается лишь одно единственное значение IDF.

$$tf(t, d) = \frac{n_t}{N} \quad (1)$$

где t – слово, для которого считается коэффициент, d – текущий документ, n_t – количество слов t в документе d , N – общее количество слов в документе d .

$$idf(t, D) = \log \frac{D}{d_t} \quad (2)$$

где D – количество документов в наборе, d_t – число документов из набора D, в которых присутствует слово t.

$$tfidf(t, d, D) = tf(t, d) * idf(t, D) \quad (3)$$

Данную операцию по векторизации методом TF-IDF необходимо однократно выполнить и для базы данных ответов (словаря). Повторное вычисление TF-IDF векторов словаря необходимо только в случае изменения/удаления текущих или добавления новых ответов.

Таким образом, на данном этапе работы системы имеется множество векторов, подсчитанных с помощью метода TF-IDF, для базы данных ответов (словаря), вектор и класс пользовательского запроса.

2) Следующим этапом является попарное вычисление меры сходства между ненулевыми векторами (коэффициента Отиаи). Первым вектором выступает вектор запроса, вторым – вектор из части базы данных ответов, относящейся к классу пользовательского запроса. Такая операция выполняется для всех векторов из БД ответов, относящихся к классу запроса.

3) На основе подсчитанных коэффициентов Отиаи выбирается один вектор из базы данных ответов. Соответственно, чем больше коэффициент (ближе к 1), тем ближе в векторном пространстве лежат обрабатываемые вектора. Вектор с наибольшим коэффициентом является численным представлением конечного ответа пользователю.

4) Конечным этапом поиска ответа пользователю является интерпретация вектора с наибольшим коэффициентом Отиаи в ответ на естественном языке. Данная операция подразумевает упорядоченный метод хранения векторов TF-IDF для ответов и выполняется путём сопоставления индексов вектора и индекса строки на естественном языке.

Таким образом, алгоритм поиска ответа на пользовательский запрос можно представить в виде блок-схемы (рис. 5).

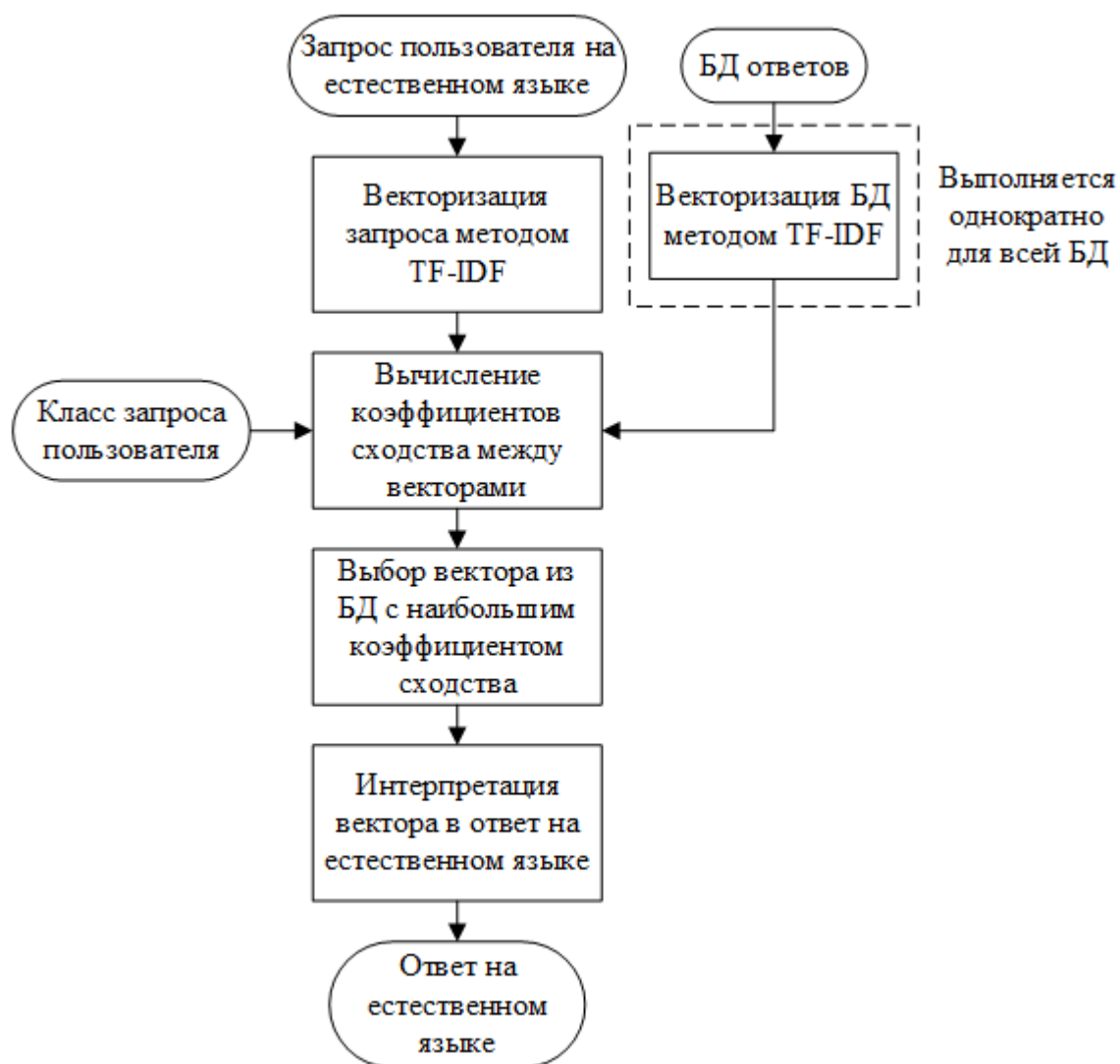


Рисунок 5. Алгоритм поиска ответа на пользовательский запрос.

3.6 Разработка словаря ответов

Ключевым звеном в разработке системы с естественно-языковым интерфейсом по выборочной (поисковой) модели является база данных (словарь) ответов на пользовательские запросы. Согласно техническому заданию, разрабатываемый словарь должен удовлетворять запросам поиска по тематикам, связанным с НГТУ им. Р. Е. Алексеева.

В ходе рассмотрения наиболее важных аспектов жизнедеятельности, связанных с ВУЗом, с которыми сталкиваются абитуриенты и студенты младших курсов, было определено 15 различных тем (интентов), а именно: банкоматы, библиотеки, второй отдел, СОЛ «Ждановец», история НГТУ, медицинская служба, общежития, парковки, питание, профком, расписание, РСМ НГТУ, структура НГТУ, студенческий клуб, транспорт. Каждая тема включает в себя различное число ответов на вопросы.

В качестве формата хранения данных был выбран формат «JSON». Данный формат был выбран исходя из удобства обработки данных, а также из-за отсутствия необходимости в разрабатываемой системе полновесной базы данных.

Пример структуры словаря приведён ниже:

```
{"tag": "Библиотеки",
```

```
  "patterns": ["Библиотека 2 корпуса, контакты: Абонемент и читальный зал научной литературы (аудитория 2303), Читальный зал учебной литературы (аудитория 2202), телефон: +7 (831) 436-17-34, email: library@nntu.ru", "Библиотека 6 корпуса, контакты: Абонементы для студентов младших и старших курсов (аудитория 6116, аудитория 6270), Абонемент для студентов заочной формы обучения (аудитория 6270), Читальный зал учебной литературы открытого доступа (аудитория 6162), Зал электронных ресурсов (аудитория 6119), телефон: +7 (831) 257-86-54, email: lib_6115@nntu.ru", "Центр культуры и чтения: абонемент художественной литературы (аудитория 6122), выставочный зал (аудитория 6123), территория свободного общения (аудитория 6117)"]
},
```

```
{"tag": "Медицинская_служба",
```

```
  "patterns": ["Медицинский кабинет в 1 корпусе: Врач: Смирнова Маргарита Николаевна, телефон: 436-03-91, время работы: с 8:30 до 15:00, кроме субботы и воскресенья", "Медицинский кабинет в 6 корпусе: Врач: Гущина Галина Ивановна, телефон: 257-86-64, время работы: с 8:30 до 15:00, кроме субботы и воскресенья",
```

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						24
Изм	Лист	№ докум.	Подп.	Дата		

"Медицинский кабинет в 1 общежитии: Врач: Санаева Любовь Викторовна, телефон: 433-52-04, время работы: с 9:00 до 16:00, кроме субботы и воскресенья"]

},

{"tag": "Второй_отдел",

"patterns": ["Второй отдел контакты: 6 корпус, аудитории 6204, 6207, 6208, 6233, телефоны: +7 (831) 257-86-69, +7 (831) 257-86-68, +7 (831) 201-04-19"]

},

Информация в словаре сформирована таким образом, чтобы ответы всегда начинались с заголовка по заданной тематике.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						25
Изм	Лист	№ докум.	Подп.	Дата		

4. Разработка программных средств

4.1 Модуль предварительной обработки

В пункте 3.2 был определён порядок предварительной обработки запроса. Предварительная обработка включает в себя:

- 1) Токенизацию;
- 2) Стемминг;
- 3) Удаление «стоп» слов.

Также предварительная обработка однократно выполняется над словарём ответов. В результате получается список токенизированных стемм слов для всего словаря и для пользовательского запроса.

Рассмотрим подробнее алгоритм предварительной обработки обработки:

Стеммер:

```
stemmer = SnowballStemmer("russian")
```

В качестве стеммера, то есть алгоритма стемматизации был выбран стеммер «SnowballStemmer», который включает в себя поддержку 25 естественных языков, в том числе, поддержку русского языка.

Далее создадим словарь (здесь «словарь» - это объект, описанный на языке Python) пунктуационных знаков, которые необходимо удалить из базы данных ответов и запроса.
`remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)`

Оба объекта создаются в глобальной области видимости программы и будут доступны для использования во всех ниже описанных функциях.

Далее выполняется однократный процесс токенизации на предложения словаря ответов, включающий в себя приведение словаря к нижнему регистру:

```
with open(str(result_class) + '.txt', 'r', encoding='utf8', errors='ignore') as fin:
```

```
    raw = fin.read().lower()
```

```
sent_tokens = nltk.sent_tokenize(raw, language="russian")
```

Ниже описаны функции, позволяющие выполнить токенизацию по словам пользовательского запроса и словаря ответов, а также удаление пунктуационных знаков, которые были описаны в словаре «remove_punct_dict» и «стоп» слов.

```
def StemTokens(tokens):
```

```
    return [stemmer.stem(token) for token in tokens]
```

```
def StemNormalize(text):
```

```
    return StemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))
```

Функция «StemNormalize» принимает на вход список, состоящий из предложений словаря ответов или предложения пользовательского запроса. Выполняет удаление пунктуационных знаков и понижение регистра, а также вызывает вспомогательную функцию «StemTokens», которая выполняет токенизацию предложений на слова. Таким образом, на выходе функции «StemNormalize» получается список токенизированных стемм слов.

Процесс удаления «стоп» слов выполняется с помощью функции «ignore_words», которая пересобирает имеющийся словарь стемм, удаляя при этом стеммы вводных слов, союзов, предлогов, частиц и т.п. Полный исходный код данных функций на языке программирования Python представлен в приложении.

Ниже приведён пример списка токенизированных стемм слов из словаря ответов:

'обучен', 'общежит', 'общен', 'основа', 'отдел', 'отделен', 'открыт', 'пар', 'парковк', 'перв', 'печер', 'питан', 'площад', 'подач', 'политехническ', 'понеделник', 'председател', 'прежн', 'приемн', 'прикладн', 'программирован', 'пропуск', 'проректор', 'профком', 'работ', 'радиосистем', 'радиоэлектроник', 'радист', 'расписан', 'ректор', 'ресурс', 'рсм', 'руководител', 'сайт', 'свободн', 'систем', 'специалист', 'справок', 'сред', 'старш', 'столов', 'структур', 'студент', 'студенческ'.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						27
Изм	Лист	№ докум.	Подп.	Дата		

4.2 Модуль вычисления признакового описания

Вычисление признакового описания, также, как и предварительная обработка, однократно выполняется для словаря ответов – это необходимо для дальнейшего обучения нейронной сети, а также многократно для каждого пользовательского запроса.

Процесс вычисления признакового описания – это процесс векторизации методом Bag of Words (рис. 6)

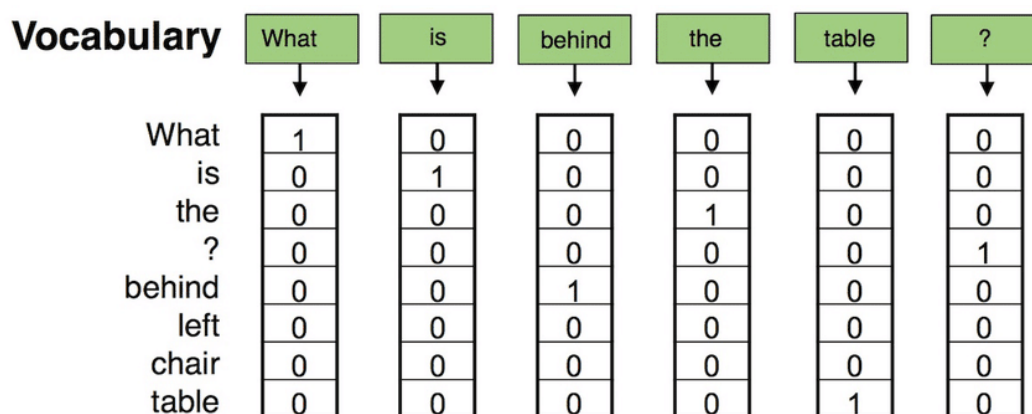


Рисунок 6. Визуализация принципа векторизации методом Bag of Words.

Рассмотрим часть функции вычисления признакового описания, описывающую непосредственно процесс векторизации ответов из словаря и их классов:

Создаём тренировочный набор (данные вектора затем понадобятся при обучении модели)

training = []

Пустой массив по количеству классов

output_empty = [0] * len(classes)

Тренировочный набор данных

for doc in documents:

 bag = []

 # Берём слова (без классов) из сформированных пар

 pattern_words = doc[0]

 # Формируем массив для стемм: 1 - если стемма присутствует в словаре, 0 - если нет

 for w in words:

 bag.append(1) if w in pattern_words else bag.append(0)

 # Формируем массив для классов:

 # 1 ставится в то место, индекс которого соответствует, обрабатываемому классу

 # остальные – 0

```

output_row = list(output_empty)
output_row[classes.index(doc[1])] = 1
# Тренировочный набор
training.append([bag, output_row])

```

В описанном выше примере кода список «documents» – это список пар по типу: ответ, токенизированный на стеммы – класс, к которому принадлежит ответ. В итоге образуются векторизованные ответы из словаря и векторизованные классы. Данные вектора необходимы для дальнейшего обучения модели.

Рассмотрим функцию векторизации пользовательского запроса:

```

def bow(sentence, vocabulary):
    sentence_words = clean_up_sentence(sentence)
    bag_words = [0] * len(vocabulary)
    for s in sentence_words:
        for i, word in enumerate(vocabulary):
            if word == s:
                bag_words[i] = 1
    return np.array(bag_words)

```

Данная функция принимает на вход два списка. Первый список – это токенизированные стеммы слов запроса пользователя, второй – также токенизированные стеммы слов из словаря ответов. Непосредственный процесс векторизации такой же, как и в случае векторизации ответов и классов ответов. На выходе получается вектор, размерность которого равна длине словаря.

4.3 Модуль классификации запроса

Следующим важнейшим этапом является реализация модели нейронной сети и её обучение. Для обучения нейронной сети потребуются те вектора, которые были созданы на предыдущем этапе реализации системы (векторизованные ответы из словаря).

Рассмотрим часть основной функции создания и обучения модели нейронной сети для классификации пользовательских запросов:

```
# Тренировочный набор (процесс создания тренировочного набора описан в пункте 4.2)
training.append([bag, output_row])

# Случайно перемешиваем тренировочный набор
random.shuffle(training)
training = np.array(training)

# Данные для нейронной сети. X - закодированные паттерны, Y - закодированные классы
train_x = list(training[:, 0])
train_y = list(training[:, 1])

X_train, X_test, Y_train, Y_test = sk.model_selection.train_test_split(train_x, train_y,
test_size=0.2, random_state=42)

# Сохраняем выборку для тестов
with open('X_test.pickle', 'wb') as f:
    pickle.dump(X_test, f)

with open('Y_test.pickle', 'wb') as f:
    pickle.dump(Y_test, f)
```

После подготовки данных для обучения необходимо создать каркас самой нейронной сети.

```
# Создаём трёхуровневую модель:

# Первый уровень - 128 нейронов, второй уровень - 64 нейрона
# третий уровень - количество нейронов = количество классов
model = Sequential()
model.add(Dense(128, input_shape=(len(X_train[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
```


Далее, выполняем компиляцию модели:

```
# Стохастический градиентный спуск с ускоренным градиентом Нестерова
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

После создания каркаса нейронной сети, модель необходимо обучить на заготовленных данных.

```
# Обучение модели
```

```
history_model = model.fit(np.array(X_train), np.array(Y_train), epochs=200, batch_size=5,
verbose=1, validation_data=(np.array(X_test), np.array(Y_test)))
```

Обучение происходит 200 эпох.

```
# Сохранение модели
```

```
model.save('my_model.h5')
```

Код функции визуализации данных «history» описан в приложении.

```
# Строим графики обучения
```

```
history(history_model)
```

Результаты обучения модели представлены на рисунке 7.

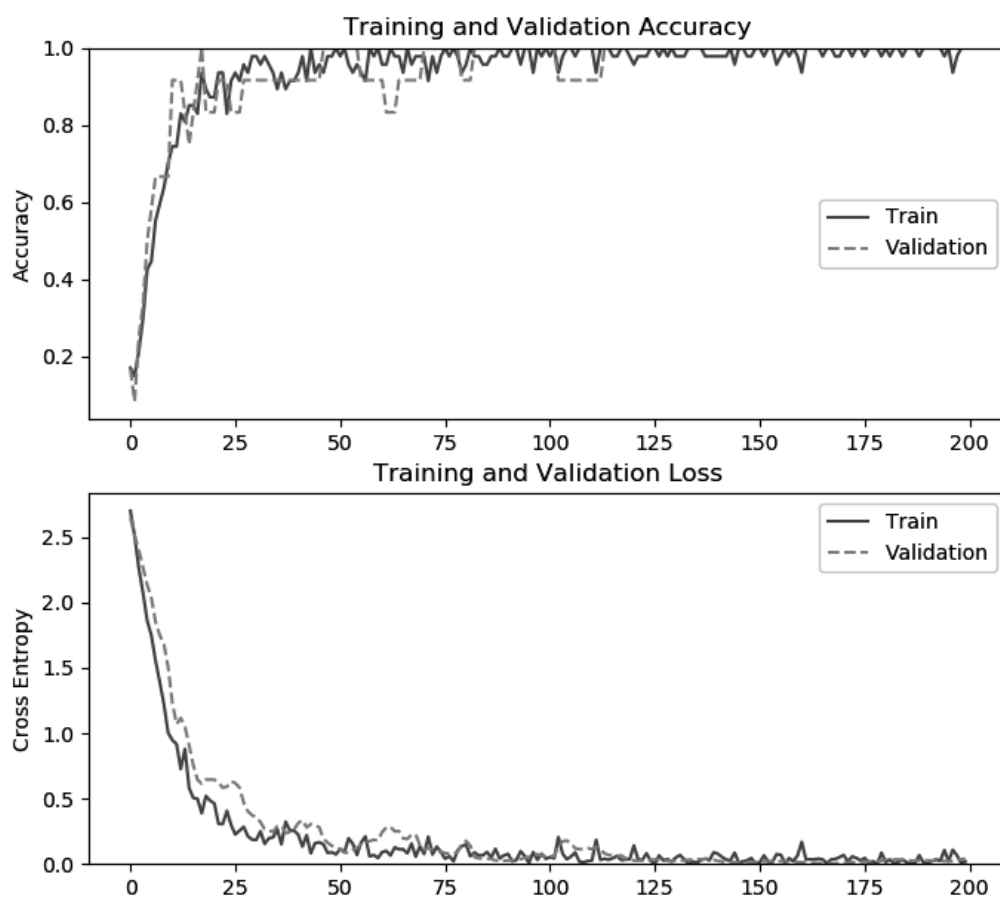


Рисунок 7. Зависимость значений точности и ошибки от количества эпох.

Как видно из графиков (рис. 7), после прохождения 200 эпох точность достигла значений, превышающих 90%, в то время как значения ошибки колеблются около значений близких к нулю. Следовательно, модель обучена успешно, переобучения модели не произошло.

Непосредственная классификация пользовательского запроса происходит следующим образом (представлена часть функции классификации):

Происходит загрузка словаря ответов (прошедшего предварительную обработку), а также списка классов. Список классов необходим для дальнейшего сопоставления выхода модели (вероятности) с названиями классов.

```
with open('words.pickle', 'rb') as f:
```

```
    words = pickle.load(f)
```

```
with open('classes.pickle', 'rb') as f:
```

```
    classes = pickle.load(f)
```

После загрузки словарей происходит векторизация методом Bag of Words пользовательского запроса.

```
test = bow(sys.argv[2], words)
```

```
inputvar = pd.DataFrame([test], dtype=float, index=['input'])
```

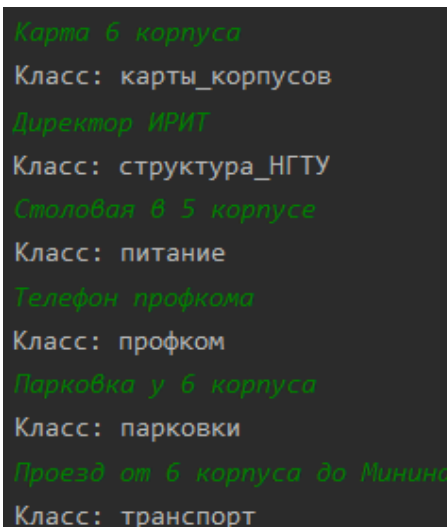
Далее, следует загрузка обученной модели и её применение.

```
model = keras.models.load_model('my_model.h5')
```

```
print(model.predict(inputvar))
```

```
print(classes)
```

Пример работы нейронной сети по классификации пользовательских запросов представлен на рисунке 8.



```
Карта 6 корпуса
Класс: карты_корпусов
Директор ИРИТ
Класс: структура_НГТУ
Столовая в 5 корпусе
Класс: питание
Телефон профкома
Класс: профком
Парковка у 6 корпуса
Класс: парковки
Проезд от 6 корпуса до Минина
Класс: транспорт
```

Рисунок 8. Результаты работы обученной модели.

Таким образом, на выходе модели получается список вероятностей принадлежности пользовательского запроса к каждому классу. Из данного списка выбирается наибольшая вероятность и происходит сопоставление этой вероятности с названием класса на естественном языке, это необходимо для дальнейшего поиска ответа в заданном классе.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						33
Изм	Лист	№ докум.	Подп.	Дата		

4.4 Модуль поиска ответа на запрос

Последним этапом формирования конечного ответа пользователю на его запрос является этап поиска данного ответа в базе данных ответов по классу, определенному на предыдущем шаге (пункт 4.3).

Данный модуль выполняет вычисление TF-IDF вектора для пользовательского запроса. Далее происходит вычисление косинусного сходства между данным вектором и набором векторов из базы данных ответов. Набор векторов для БД ответов вычисляется однократно и требует пересчёта только в случае изменения базы.

Рассмотрим алгоритм формирования конечного ответа на пользовательский запрос:

```
flag = True
```

```
while flag:
```

```
    user_response = input()
```

```
    Происходит приведение запроса к нижнему регистру.
```

```
    user_response = user_response.lower()
```

```
    if user_response == 'спасибо' or user_response == "пока" \
        or user_response == "досвидания":
```

```
        flag = False
```

```
        print("Досвидания, хорошего дня =)")
```

```
    else:
```

Далее, происходит предварительная обработка, векторизация и классификация пользовательского запроса с помощью обученной модели.

```
    with open('words.pickle', 'rb') as f:
```

```
        words = pickle.load(f)
```

```
    with open('classes.pickle', 'rb') as f:
```

```
        classes = pickle.load(f)
```

```
    user_response = SnowballStemmer.stem(user_response)
```

```
    test = bow(user_response, words)
```

```
    inputvar = pd.DataFrame([test], dtype=float, index=['input'])
```

```
    model = keras.models.load_model('my_model.h5')
```

```
    result_class_list = model.predict(inputvar)
```

```
    result_class_list = list(result_class_list[0])
```

```
    result_class_ind = result_class_list.index(max(result_class_list))
```

```
    result_class = classes[result_class_ind]
```

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						34
Изм	Лист	№ докум.	Подп.	Дата		

С помощью полученного результата классификации пользовательского запроса происходит загрузка нужного списка векторов из БД ответов.

```
with open(str(result_class) + '.pickle', 'rb') as f:  
    list_tfidf = pickle.load(f)
```

Далее, происходит вызов функции «response», описанной ниже.

```
print(response(user_response))
```

Рассмотрим пример функции вычисляющей TF-IDF вектор для пользовательского запроса и косинусного коэффициента сходства данного вектора с векторами из БД ответов:

```
def response(user_response):  
    robo_response = ""  
    TfidfVec = TfidfVectorizer(tokenizer=StemNormalize)  
    tfidf = TfidfVec.fit_transform(user_response)  
    list_tfidf.append(tfidf)  
    vals = cosine_similarity(tfidf[-1], tfidf)  
    idx = vals.argsort()[0][-2]  
    flat = vals.flatten()  
    flat.sort()  
    req_tfidf = flat[-2]  
    if (req_tfidf == 0):  
        robo_response = robo_response + "Извините, я Вас не понимаю :(\n" \\  
            "Пожалуйста, задайте другой вопрос"  
    return robo_response  
else:  
    robo_response = robo_response + response[idx]  
    return robo_response
```

В данном примере функция «TfidfVec.fit_transform()» выполняет вычисление TF-IDF вектор для пользовательского запроса. Далее, с помощью функции «cosine_similarity()» происходит вычисление косинусного сходства между полученным вектором и векторами из БД ответов. Естественно, более похожие вектора имеют больший косинусный коэффициент сходства (близкий к 1), и наоборот, менее похожие вектора – меньший косинусный коэффициент. Последним этапом является трансляция полученного наибольшего значения косинусного коэффициента в ответ на естественном языке.

Следует заметить, что обмен запросами и ответами между пользователем и системой происходит через консоль. На следующем этапе (пункт 4.5) рассмотрен конечный вариант программной системы с использованием возможностей ПО «Telegram Messenger».

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						36
Изм	Лист	№ докум.	Подп.	Дата		

4.5 Модуль взаимодействия с ПО «Telegram Messenger»

Для удобства использования разрабатываемой программной системы было принято решение создать так называемого чат-бота на основе ПО «Telegram Messenger». Модуль чат-бота будет отвечать за принятие пользовательского запроса от сервера «Telegram Messenger», передачу текстовой информации на обработку и отправку результата обратно на сервер «Telegram Messenger» с указанием идентификатора клиента, от которого пришел данный запрос.

Для взаимодействия с клиентской частью мессенджера «Telegram Messenger» необходимо подключить библиотеку «telebot».

Рассмотрим часть программного кода, выполняющего вышеописанные функции.

```
bot = telebot.TeleBot("XXX", threaded=False)
```

где "XXX" – это уникальный идентификатор.

В первую очередь необходимо идентифицировать обработчика запросов, то есть присвоить ему некий уникальный идентификатор. Данный уникальный ID генерируется сервером «Telegram Messenger» при создании чат-бота. Он необходим для того, чтобы через клиентскую часть программы «Telegram Messenger», при включении нашего обработчика, сказать серверу «Telegram Messenger» о том, что обработчик запросов с данным ID находится по данному IP-адресу.

Далее описывается декоратор для функции «bot.message_handler», который включает в себя все методы, описанные в пунктах 4.1, 4.2, 4.4. Декоратор необходим для замены стандартного обработчика запросов нашей системой.

Приведем только заголовок данного декоратора:

```
@bot.message_handler(content_types=["text"])
```

```
def handle_text(message):
```

```
...
```

Рассмотрим формирование ответа на стандартную команду «/start», которая выполняется при запуске чат-бота.

```
if message.text == "/start":
```

```
list_help = "Assistant_v.1.0 служит для помощи абитуриентам и студентам  
младших курсов НГТУ им. Р. Е. Алексеева" \  
+ "\n" + "Данный бот ориентирован на предоставление справочной  
информации, связанной ВУЗом." \  

```

+ "\n" + "Список тем доступен по команде /help"

bot.send_message(message.from_user.id, list_help)

В данном примере, при получении команды «/start» происходит формирование информационной строки и отправление её обратно пользователю. Отправка сообщения выполняется с помощью метода «bot.send_message» и включает обязательное указание ID клиента получателя (message.from_user.id). Данный ID берётся из пользовательского запроса.

На рисунке 9 приведён пример ответа на команду «/start» в ПО «Telegram Messenger».

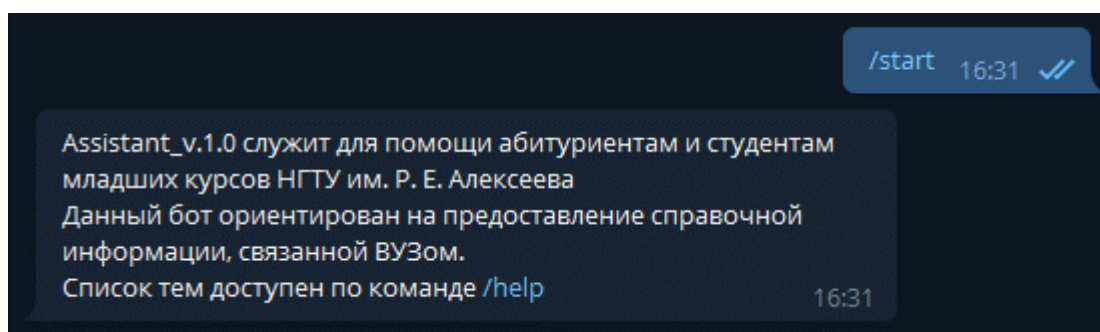


Рисунок 9. Ответ на команду «/start».

5. Тестирование системы

Необходимым этапом в создании программной системы является этап её тестирования. Наилучшим способом для проверки работоспособности программной системы с естественно-языковым интерфейсом является непосредственное её использование в реальных условиях.

Для тестирования созданной системы были привлечены студенты второго и третьего курсов обучения НГТУ им. Р. Е. Алексеева - всего 90 человек.

Требования, которые предъявлялись к персональным компьютерам (или смартфонам) студентов, согласившимся принять участие в тестировании системы:

1. Доступ в сеть Интернет;
2. Наличие установленного ПО «Telegram Messenger»;
3. Наличие учётной записи в системе «Telegram Messenger».

Для анализа выходных данных, то есть ответов на пользовательские запросы, была реализована система лог-файлов.

В лог-файлы производилась запись следующей информации:

1. Пользовательский запрос на естественном языке;
2. Класс пользовательского запроса, полученный на выходе модели;
3. Конечный ответ пользователю на естественном языке.

Каждый студент сформировал по 10 запросов, то есть, всего 900 тестовых запросов.

Результаты тестирования оказались следующими:

1. 772 запроса были верно классифицированы из них:
 - a. На 721 – дан верный ответ;
 - b. На 51 – ошибочный ответ;
2. 128 запросов были классифицированы ошибочно.

По результатам тестирования можно сделать вывод о точности работы классификатора и созданной системы в целом. Точность работы классификатора составляет около 86 процентов, точность всей системы составляет чуть более 80 процентов.

Рассмотрим примеры работы системы при различных входных данных. Для начала рассмотрим выводы системы на запросы, ответы на которые явно присутствуют в базе данных (рис. 10, 11, 12).

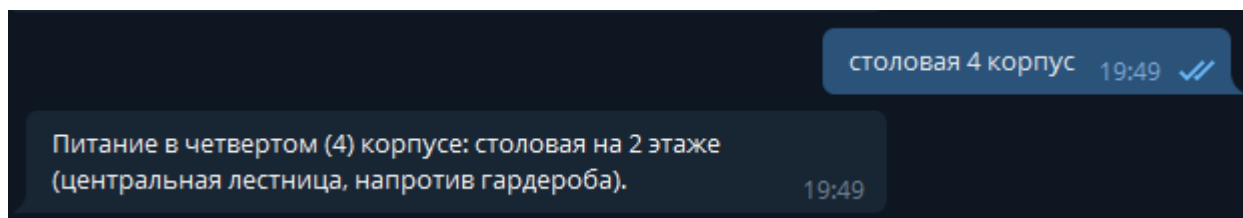


Рисунок 10. Пример работы системы, при наличии класса запроса в БД.

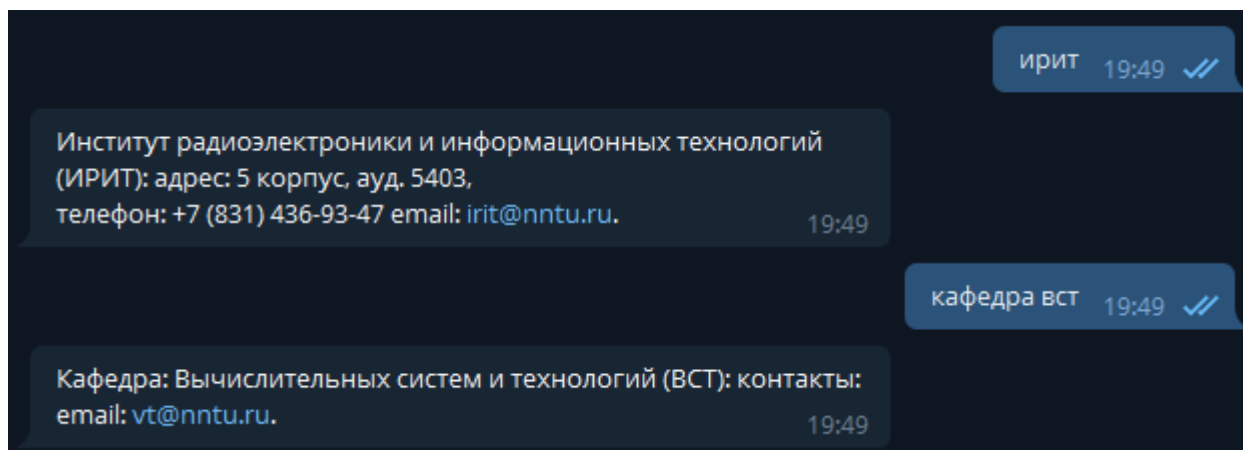


Рисунок 11. Пример работы системы, при наличии класса запроса в БД.

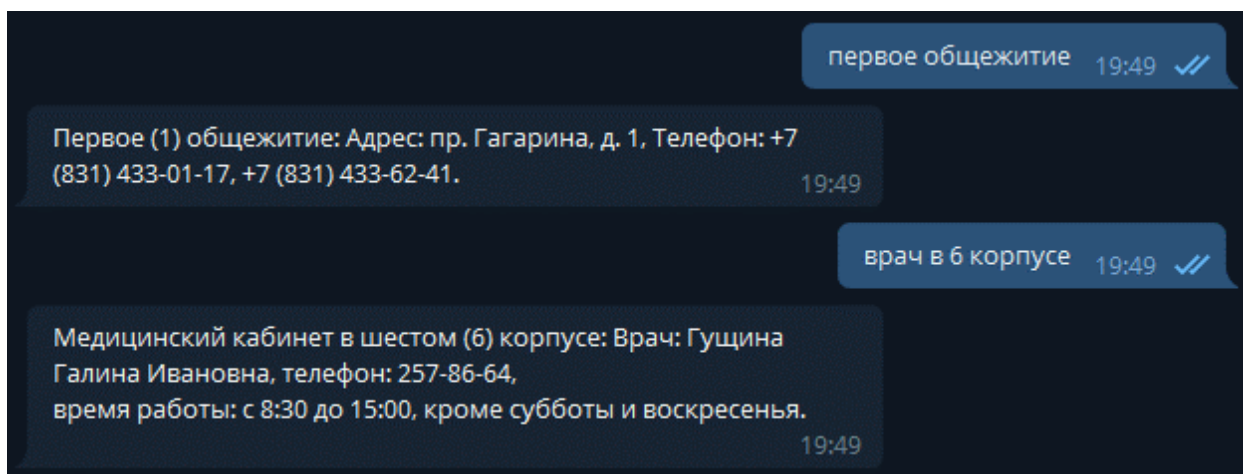


Рисунок 12. Пример работы системы, при наличии класса запроса в БД.

Как видно из рисунков 10, 11 и 12 система правильно классифицировала пользовательские запросы и нашла конечные ответы на них.

Далее, рассмотрим результаты работы системы при входных запросах, классов которых нет в базе данных ответов (рис. 13, 14).

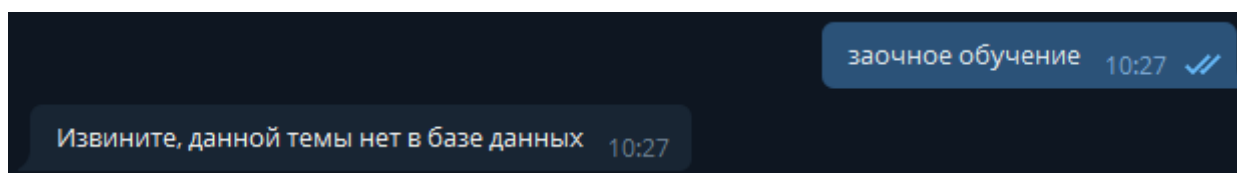


Рисунок 13. Пример работы системы, при отсутствии класса запроса в БД.

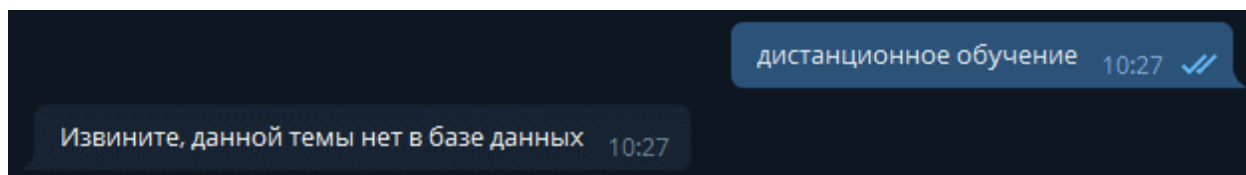


Рисунок 14. Пример работы системы, при отсутствии класса запроса в БД.

В данном случае система, как и ожидалось, не смогла определить конечный ответ в базе данных. После классификации данных запросов заведомо определился неверный класс. Далее, происходит вычисление косинусного коэффициента для определения конечного ответа внутри класса, однако, ни один из имеющихся ответов не превысил порог косинусного коэффициента (порог равен $1/4$ или $\pm 75^\circ$ и был подобран эмпирическим методом), и как следствие, на выходе системы формируется сообщение об отсутствии данной темы в базе данных.

Заключение

В данной выпускной квалификационной работе была спроектирована и реализована программная система с естественно-языковым интерфейсом для помощи абитуриентам и студентам младших курсов НГТУ им. Р. Е. Алексеева.

В созданной системе реализованы подсистема предварительной обработки пользовательских запросов, включающая в себя токенизацию, стемминг и удаление «стоп» слов, подсистема вычисления признакового описания, подсистема классификации на основе нейронной сети и подсистема поиска конечного ответа в БД, основанная на вычислении tf-idf и косинусного коэффициентов.

Настоящая программная система была интегрирована с ПО «Telegram Messenger», что позволило использовать готовый пользовательский интерфейс и клиент-серверную структуру.

Перспектива развития данной программной системы заключается в следующем:

1. Создание подсистемы обработки голосовых запросов;
2. Создание подсистемы дообучения модели нейронной сети;
3. Внедрение многопоточной обработки пользовательских запросов.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
Изм	Лист	№ докум.	Подп.	Дата		42

Список литературы

1. Steven Bird, Ewan Klein, Edward Loper. Natural Language Processing with Python // O'Reilly Media, 2009.
2. Jones K. S. A statistical interpretation of term specificity and its application in retrieval // MCB University Press, 2004. – V. 60. – N. 5. – P. 493-502.
3. Дж Солтон. Динамические библиотечно-поисковые системы // Мир, 1979.
4. Lovins, Julie Beth. Development of a Stemming Algorithm // Mechanical Translation and Computational Linguistics, 1968. – Т. 11.
5. Маннинг К., Рагхаван П., Шютце Х. Введение в информационный поиск. — Вильямс, 2011. – 512 с.
6. Cheetam A.H., Hazel J.E. Binary similarity coefficients // J. Paleontology, 1969. V. 43. – N. 5. – P. 1130—1136.
7. Youngjoong Ko (2012). «A study of term weighting schemes using class information for text classification». SIGIR'12. ACM.
8. Sepp Hochreiter; Jürgen Schmidhuber. Long short-term memory // Neural Computation journal, 1997. — V. 9. – N. 8. — P. 1735—1780.

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						43
Изм	Лист	№ докум.	Подп.	Дата		

Приложение

1. Create_model.py – файл с созданием, обучением и тестированием модели:

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.optimizers import SGD

import numpy as np

import pandas as pd

import pickle

import random

import nltk

from nltk.stem.snowball import SnowballStemmer

import json

import sys

import keras

import matplotlib.pyplot as plt

import sklearn as sk

import netron

import random


def history(history_model):

    acc = history_model.history['accuracy']

    val_acc = history_model.history['val_accuracy']

    loss = history_model.history['loss']

    val_loss = history_model.history['val_loss']

    plt.figure(figsize=(8, 8))

    plt.subplot(2, 1, 1)

    plt.plot(acc, label='Train')

    plt.plot(val_acc, '--', label='Validation')

    plt.legend(loc='right')
```

					ВКР-ИГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						44
Изм	Лист	№ докум.	Подп.	Дата		

```

plt.ylabel('Accuracy')

plt.ylim([min(plt.ylim()), 1])

plt.title('Training and Validation Accuracy')

plt.subplot(2, 1, 2)

plt.plot(loss, label='Train')

plt.plot(val_loss, '--', label='Validation')

plt.legend(loc='upper right')

plt.ylabel('Cross Entropy')

plt.ylim([0, max(plt.ylim())])

plt.title('Training and Validation Loss')

plt.savefig('FIGURE_NAME')

plt.show()

print("Success")

# Стеммер
stemmer = SnowballStemmer("russian")

def clean_up_sentence(sentence):

    sentence_words = nltk.word_tokenize(sentence)

    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]

    return sentence_words

def bow(sentence, vocabulary):

    sentence_words = clean_up_sentence(sentence)

    bag_words = [0] * len(vocabulary)

    for s in sentence_words:

        for i, word in enumerate(vocabulary):

            if word == s:

                bag_words[i] = 1

    return np.array(bag_words)

```

					<i>BKP-ИГТУ-09.03.01-(16-B-2)-002-2020(ИЗ)</i>	Лист
						45
Изм	Лист	№ докум.	Подп.	Дата		

```

if __name__ == "__main__":
    if len(sys.argv) > 1:
        if (sys.argv[1] == "обучение") or (sys.argv[1] == "Обучение"):
            # Загрузка интенгов
            with open('intents.json', encoding="utf8") as json_data:
                intents = json.load(json_data)
            # print(intents)

            # Создание словаря и предварительная обработка данных
            words = []
            classes = []
            documents = []
            ignore_words = []

            # Цикл по каждому паттерну в интенгах
            for intent in intents['intents']:
                for pattern in intent['patterns']:
                    # Разделяем паттерн на слова (токены)
                    w = nltk.word_tokenize(pattern)

                    # Формируем список слов
                    words.extend(w)

                    # Формируем пары (слова из паттерна - класс интенга, к которому
                    принадлежит этот паттерн)
                    documents.append((w, intent['tag']))

                    # Формируем список классов
                    if intent['tag'] not in classes:
                        classes.append(intent['tag'])

            #print(documents)

            # Проводим стемминг над списком слов, убираем повторы и слова-пропуски
            words = [stemmer.stem(w.lower()) for w in words if w not in ignore_words]

```

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						46
Изм.	Лист	№ докум.	Подп.	Дата		


```

words = sorted(list(set(words)))

# Сохраняем словарь
with open('words.pickle', 'wb') as f:
    pickle.dump(words, f)

# Сортируем список классов
classes = sorted(list(set(classes)))

# Сохраняем список классов
with open('classes.pickle', 'wb') as f:
    pickle.dump(classes, f)

# print (len(documents), "пар")
# print ("Количество классов:",len(classes), "\n", "Классы:", classes)
# Длина словаря
#print ("Длина словаря:", len(words), "\n", "Стемы слов:", words)

# Создаём тренировочный набор
training = []

# Пустой массив по количеству классов
output_empty = [0] * len(classes)

# Тренировочный набор данных
for doc in documents:
    bag = []
    # Берём слова (без классов) из сформированных пар
    pattern_words = doc[0]
    # Проводим над ними стемминг
    pattern_words = [stemmer.stem(word.lower()) for word in pattern_words]
    # Формируем массив для стемм: 1 - если стемма присутствует в словаре, 0 -
если нет
    for w in words:
        bag.append(1) if w in pattern_words else bag.append(0)

```

```

# Формируем массив для классов:

# 1 ставится в то место, индекс которого соответствует, обрабатываемому
классу

# остальные - 0

output_row = list(output_empty)

output_row[classes.index(doc[1])] = 1

# Тренировочный набор

training.append([bag, output_row])

# Случайно перемешиваем тренировочный набор

random.shuffle(training)

training = np.array(training)

# Данные для нейронной сети. X - закодированные паттерны, Y - закодированные
классы

train_x = list(training[:, 0])

train_y = list(training[:, 1])

X_train, X_test, Y_train, Y_test = sk.model_selection.train_test_split(train_x, train_y,
test_size=0.2, random_state=42)

# Сохраняем выборку для тестов

with open('X_test.pickle', 'wb') as f:

    pickle.dump(X_test, f)

with open('Y_test.pickle', 'wb') as f:

    pickle.dump(Y_test, f)

# Создаём трёхуровневую модель:

# Первый уровень - 128 нейронов, второй уровень - 64 нейрона

# третий уровень - количество нейронов = количеству классов

model = Sequential()

model.add(Dense(128, input_shape=(len(X_train[0]),), activation='relu'))

model.add(Dropout(0.5))

```

```

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(len(train_y[0]), activation='softmax'))


# Компилируем модель

# Стохастический градиентный спуск с ускоренным градиентом Нестерова
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])

# Обучение модели

history_model = model.fit(np.array(X_train), np.array(Y_train), epochs=200,
batch_size=5, verbose=1, validation_data=(np.array(X_test), np.array(Y_test)))

# Сохранение модели

model.save('my_model.h5')

print(history_model.history.keys())

# Строим графики обучения

history(history_model)

# Визуализация нейронной сети

netron.start('D:/test/my_model.h5')


# Тестирование модели

if (sys.argv[1] == "тест") or (sys.argv[1] == "Тест"):

    with open('words.pickle', 'rb') as f:

        words = pickle.load(f)

    print(words)

    with open('classes.pickle', 'rb') as f:

        classes = pickle.load(f)

    if len(sys.argv) > 2:

        test = bow(sys.argv[2], words)

        inputvar = pd.DataFrame([test], dtype=float, index=['input'])

        model = keras.models.load_model('my_model.h5')

```

```

        #print(test)

        #print(inputvar)

        print(model.predict(inputvar))

        print(classes)

    else:

        with open('X_test.pickle', 'rb') as f:

            X_test = pickle.load(f)

        with open('Y_test.pickle', 'rb') as f:

            Y_test = pickle.load(f)

        #print(X_test[0])

        model = keras.models.load_model('my_model.h5')

        for i in range(len(X_test)):

            X_test[i] = np.array(X_test[i])

            inputvar = pd.DataFrame([X_test[i]], dtype=float, index=['input'])

            print(model.predict(inputvar))

            print(classes)

            print(Y_test[i])

    else:

        exit()

```

2. find_answer.py – файл, включающий в себя вычисление tf-idf и косинусного коэффициентов:

```

import random

import string

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

import warnings

import nltk

from nltk.stem import WordNetLemmatizer

import pandas as pd

```

					ВКР-ИГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						50
Изм	Лист	№ докум.	Подп.	Дата		

```

import keras

import pickle

import sys

from nltk.stem.snowball import SnowballStemmer

import numpy as np


# СТЕММЕР
stemmer = SnowballStemmer("russian")

warnings.filterwarnings('ignore')

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)


def clean_up_sentence(sentence):

    sentence_words = nltk.word_tokenize(sentence)

    sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]

    return sentence_words


def bow(sentence, vocabulary):

    sentence_words = clean_up_sentence(sentence)

    bag_words = [0] * len(vocabulary)

    for s in sentence_words:

        for i, word in enumerate(vocabulary):

            if word == s:

                bag_words[i] = 1

    return np.array(bag_words)


def greeting(sentence):

    for word in sentence.split():

        if word.lower() in GREETING_INPUTS:

            return random.choice(GREETING_RESPONSES)

```

					БКР-ИГТ-09.03.01-(16-В-2)-002-2020(ИЗ)	Лист
						51
Изм	Лист	№ докум.	Подп.	Дата		

```

def response(user_response):
    robo_response = ""
    sent_tokens.append(user_response)
    TfIdfVec = TfIdfVectorizer(tokenizer=LemNormalize)
    tfidf = TfIdfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    print(vals)
    idx = vals.argsort()[0][-2]
    print(idx)
    flat = vals.flatten()
    flat.sort()
    print(flat)
    req_tfidf = flat[-2]
    print(req_tfidf)
    if (req_tfidf == 0):
        #robo_response = robo_response + "Извините, я Вас не понимаю :(\n" \
        #                                #"Пожалуйста, задайте другой вопрос"
        return robo_response
    else:
        robo_response = robo_response + sent_tokens[idx]
        return robo_response

# Токенизация входного предложения
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]
def LemNormalize(text):
    return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))

```

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						52
Изм	Лист	№ докум.	Подп.	Дата		

```

# Генерация приветствия
GREETING_INPUTS = ("привет", "добрый день", "здравствуй",)
GREETING_RESPONSES = ["Добрый день", "Здравствуйте", "Привет"]

flag = True

while flag:

    user_response = input()

    user_response = user_response.lower()

    if user_response == 'спасибо' or user_response == "пока" or user_response == "досвидания":

        flag = False

        print("Досвидания, хорошего дня =)")

    else:

        if greeting(user_response) is not None:

            print(greeting(user_response))

        else:

            with open('words.pickle', 'rb') as f:

                words = pickle.load(f)

            with open('classes.pickle', 'rb') as f:

                classes = pickle.load(f)

            test = bow(user_response, words)

            inputvar = pd.DataFrame([test], dtype=float, index=['input'])

            model = keras.models.load_model('my_model.h5')

            result_class_list = model.predict(inputvar)

            result_class_list = list(result_class_list[0])

            #print(result_class_list)

            result_class_ind = result_class_list.index(max(result_class_list))

            #print(result_class_ind)

            result_class = classes[result_class_ind]

            #print(model.predict(inputvar))

```

```

#print(classes)

print("Класс: " + result_class)

#print(result_class)

# Чтение базовой информации

with open(str(result_class) + '.txt', 'r', encoding='utf8', errors='ignore') as fin:

    raw = fin.read().lower()

# Токенизация базовой информации

sent_tokens = nltk.sent_tokenize(raw, language="russian") # converts to list of
sentences

# word_tokens = nltk.word_tokenize(raw, language="russian") # converts to list of
words

print(response(user_response))

sent_tokens.remove(user_response)

```

3. telegram_bot.py – файл обобщающей системы.

```

import telebot

import random

import string

from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.metrics.pairwise import cosine_similarity

import warnings

import nltk

from nltk.stem import WordNetLemmatizer

import pandas as pd

import keras

import pickle

from nltk.stem.snowball import SnowballStemmer

import numpy as np

```

					ВКР-ИГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						54
Изм.	Лист	№ докум.	Подп.	Дата		


```

import re

bot = telebot.TeleBot("972943152:AAG9g6OgMyDhtVoESZ_0it0Pbog_SAOKFJU",
threaded=False)

# СТЕММЕР
stemmer = SnowballStemmer("russian")

warnings.filterwarnings('ignore')

remove_punct_dict = dict((ord(punct), None) for punct in string.punctuation)

with open('words.pickle', 'rb') as f:
    words = pickle.load(f)

with open('classes.pickle', 'rb') as f:
    classes = pickle.load(f)

model = keras.models.load_model('my_model.h5')

@bot.message_handler(content_types=["text"])
def handle_text(message):
    def clean_up_sentence(sentence):
        sentence_words = nltk.word_tokenize(sentence)
        sentence_words = [stemmer.stem(word.lower()) for word in sentence_words]
        return sentence_words

    def bow(sentence, vocabulary):
        sentence_words = clean_up_sentence(sentence)
        bag_words = [0] * len(vocabulary)
        for s in sentence_words:
            for i, word in enumerate(vocabulary):
                if word == s:
                    bag_words[i] = 1
        return np.array(bag_words)

```

					БКР-ИГТ-09.03.01-(16-В-2)-002-2020(ИЗ)	Лист
						55
Изм.	Лист	№ докум.	Подп.	Дата		

```

def greeting(sentence):
    for word in sentence.split():
        if word.lower() in GREETING_INPUTS:
            return random.choice(GREETING_RESPONSES)

def response(user_response):
    robo_response = "
    sent_tokens.append(user_response)
    TfidfVec = TfidfVectorizer(tokenizer=LemNormalize)
    tfidf = TfidfVec.fit_transform(sent_tokens)
    vals = cosine_similarity(tfidf[-1], tfidf)
    idx = vals.argsort()[0][-2]
    flat = vals.flatten()
    flat.sort()
    req_tfidf = flat[-2]
    if req_tfidf == 0:
        robo_response = robo_response + "Извините, я Вас не понимаю :(\n" \
            "Пожалуйста, введите другой запрос"
        return robo_response
    else:
        robo_response = robo_response + sent_tokens[idx]
        return robo_response

# Токенизация входного предложения
def LemTokens(tokens):
    return [lemmer.lemmatize(token) for token in tokens]

def LemNormalize(text):

```

					ВКР-НГТУ-09.03.01-(16-В-2)-002-2020(ПЗ)	Лист
						56
Изм	Лист	№ докум.	Подп.	Дата		

```

return LemTokens(nltk.word_tokenize(text.lower().translate(remove_punct_dict)))

if message.text == "/start":

    list_help = "Assistant_v.1.0 служит для помощи абитуриентам и студентам младших
курсов НГТУ им. Р. Е. Алексеева" \

        + "\n" + "Данный бот ориентирован на предоставление справочной
информации, связанной ВУЗом." \

        + "\n" + "Список тем доступен по команде /help"

    bot.send_message(message.from_user.id, list_help)

elif (message.text == "/help") or (message.text == "help") or (message.text == "Help"):

    list_class = "Список тем для поиска справочной информации:" + "\n" + "Банкоматы
(/ATMs)" + "\n" + "Библиотеки (/libraries)" \

        + "\n" + "Второй отдел (/2_department)" + "\n" + "Ждановец (/zhdanovets)" +
"\n" + "История НГТУ (/history)" + "\n" + \

        "Медицинская служба (/ambulance)" + "\n" + "Общежития (/dormitory)" +
"\n" + "Парковки (/parking)" + "\n" + "Питание (/food)" + "\n" + \

        "Профком (/trade_union)" + "\n" + "Карты корпусов (/maps)" + "\n" + "PCM
(/ryu)" + "\n" + "Структура ВУЗа (/university_structure)" + "\n" + \

        "Студенческий клуб (/student_club)" + "\n" + "Транспорт (/transport)"

    bot.send_message(message.from_user.id, list_class)

else:

    user_response = message.text

    # Запись пользовательского запроса в файл (для отладки)

    file_in = open('in.txt', 'a')

    file_in.write(user_response + '\n')

    user_response = user_response.lower()

    if user_response == 'спасибо' or user_response == "пока" or user_response == "до
свидания":

        bot.send_message(message.from_user.id, "До свидания, хорошего дня =)")

    else:

        if greeting(user_response) is not None:

            bot.send_message(message.from_user.id, greeting(user_response))

        else:

```

```

test = bow(user_response, words)

inputvar = pd.DataFrame([test], dtype=float, index=['input'])

result_class_list = model.predict(inputvar)
result_class_list = list(result_class_list[0])
result_class_ind = result_class_list.index(max(result_class_list))
result_class = classes[result_class_ind]

# bot.send_message(message.from_user.id, result_class)

# Чтение базовой информации
with open(str(result_class) + '.txt', 'r', encoding='utf8', errors='ignore') as fin:
    raw = fin.read().lower()

# Токенизация базовой информации
sent_tokens = nltk.sent_tokenize(raw, language="russian") # converts to list of
sentences

res = response(user_response)
print(result_class)

if result_class == "карты_корпусов":
    if "1" in res:
        doc1 = open("D:/test/1.bmp", 'rb')
        bot.send_photo(message.from_user.id, doc1, res)
        sent_tokens.remove(user_response)
    if "2" in res:
        doc2 = open("D:/test/2.bmp", 'rb')
        bot.send_photo(message.from_user.id, doc2, res)
        sent_tokens.remove(user_response)
    if "3" in res:
        doc3 = open("D:/test/3.bmp", 'rb')
        bot.send_photo(message.from_user.id, doc3, res)

```

```

        sent_tokens.remove(user_response)

    if "4" in res:

        doc4 = open("D:/test/4.bmp", 'rb')

        bot.send_photo(message.from_user.id, doc4, res)

        sent_tokens.remove(user_response)

    if "5" in res:

        doc5 = open("D:/test/5.bmp", 'rb')

        bot.send_photo(message.from_user.id, doc5, res)

        sent_tokens.remove(user_response)

    if "6" in res:

        doc6 = open("D:/test/6.bmp", 'rb')

        bot.send_photo(message.from_user.id, doc6, res)

        sent_tokens.remove(user_response)

    if ("аудитория" in res) or ("класс" in res) or ("кабинет" in res):

        num = re.findall(r'\d{4}', user_response)

        num1 = num[0][0] + " корпус"

        num2 = num[0][1] + " этаж"

        bot.send_message(message.from_user.id, num1)

        bot.send_message(message.from_user.id, num2)

    else:

        bot.send_message(message.from_user.id, res)

        sent_tokens.remove(user_response)

        # Запись ответа в файл (для отладки)

        file_out = open('out.txt', 'a')

        file_out.write(res + "\n")

        file_in.close()

        file_out.close()

bot.polling(none_stop=True, interval=0)

```