**РОСЖЕЛДОР**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ**

**ВЫСШЕГО ОБРАЗОВАНИЯ**

**СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ПУТЕЙ СООБЩЕНИЯ (СГУПС)**

|  |  |  |  |
| --- | --- | --- | --- |
| **К защите:** |  |  | |
| **Заведующий кафедрой** | **Информационные** | |
| **технологии транспорта** | | |
|  | д-р техн. наук, проф. | |
|  |  | В. И. Хабаров | |
| *подпись* |  | *инициалы, фамилия* | |
|  |  |  | |
| *дата* |  |  | |

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА**

**(МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ)**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Тема:** | Разработка классификатора обращений граждан на | | | |
| основе нейросетей | | | | |
|  |  | МД.МИСТ.01.2022 |  |  |
|  |  | *шифр документа* |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | **Выполнил** |  |  |  | **Руководитель** |
|  |  | М. А. Беков |  |  |  | канд. техн. наук, доц.  С. П. Сарычев |
| *подпись* |  | *инициалы, фамилия* |  | *подпись* |  | *инициалы, фамилия* |
|  |  |  |  |  |  |  |
| *дата* |  |  |  | *дата* |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Нормоконтролер работы** |  |  |  | ст. преп.  Т. А. Распопина |
|  |  | *подпись* |  | *инициалы, фамилия* |
|  |  | *дата* |  |

**2022 г.**

**СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**ПУТЕЙ СООБЩЕНИЯ (СГУПС)**

|  |  |
| --- | --- |
| **Факультет:** | Бизнес-информатика |
| **Кафедра:** | Информационные технологии транспорта |
| **Направление:** | 09.04.02 «Информационные системы и технологии» |
| **Профиль:** | Интеллектуальные транспортные системы |

|  |  |  |
| --- | --- | --- |
| **УТВЕРЖДАЮ**  Заведующий кафедрой |  | «Информационные технологии транспорта»  Д-р техн. наук, проф.  В.И. Хабаров |
|  |  | *инициалы, фамилия* |
| «26» мая 2022 г. |  |  |

**ЗАДАНИЕ**

**на выполнение магистерской диссертации**

|  |  |  |  |
| --- | --- | --- | --- |
| студенту | Бекову Михаилу Аркадьевичу | | |
|  |  | | |
| 1. Тема «Разработка классификатора обращений граждан на основе нейросетей» утверждена приказом №180/c от «16» мая 2022 г. | | | |
| 2. Задание выдано «20» февраля 2022 г. | | | |
| 3. Срок сдачи законченной работы на кафедру «25» июня 2022 г. | | | |
| 4. Исходные данные: данные, полученные в ходе прохождения преддипломной практики | | | |
| 5. Содержание расчетно-пояснительной записки | | | |
| Наименование разделов и вопросов | | Примерное количество страниц | График (сроки) выполнения |
| Введение | | 2 | 10.05.2022 |
| Описание предметной области | | 14 | 10.05.2022 |
| Теоретический анализ задачи | | 14 | 13.05.2022 |
| Практическая реализация | | 33 | 28.05.2022 |
| Заключение | | 1 | 03.06.2022 |

6.Содержание и объемы графической части

|  |  |  |
| --- | --- | --- |
| Наименование графического документа (чертежа, схемы, графика) | Количество  листов  формата А1 | График  (сроки)  выполнения |
| Презентация PowerPoint | 20 | 05.06.2022 |

|  |  |  |
| --- | --- | --- |
| Руководитель |  | С. П. Сарычев |
|  | *(подпись, Фамилия, И.О.)* |  |
| Задание к использованию принял |  | М. А. Беков |
|  | *(подпись студента)* |  |

УДК 004.855.5

**АННОТАЦИЯ**

В работе 74 страницы, 58 рисунков, 1 таблица, 17 источников, 1 приложение.

Ключевые слова: *искусственный интеллект, нейросети, чат-бот, автоклассификация, Bag Of Words, Embedding, Python, Telegram.*

На рынке данное решение отсутствует во многих компаниях, из-за чего без автоматизации сотрудникам приходится выполнять однотипную работу изо дня в день. Данных людей можно перевести в другие отделы, нуждающиеся в сотрудниках, не добавляя новое штатное место и не лишая их работы. Не исключен и фактор человеческой ошибки, в рамках которой сотрудник может по ошибке или не знанию отправить обращение не в тот отдел, достаточно обученный ИИ решает и эту проблему.

**ABSTRACT**

The work contains 74 pages, 58 figures, 1 table, 17 sources, 1 addition.

Key words: *artificial intelligence, neural networks, chatbot, auto-classification, Bag Of Words, Embedding, Python, Telegram.*

In the market, this solution is not available in many companies, which is why employees have to do the same type of work day after day without automation. These people can be transferred to other departments that need employees without adding a new full-time position and without depriving them of work. The factor of human error is also not excluded, in which an employee may mistakenly or unknowingly send an appeal to the wrong department, a sufficiently trained AI solves this problem.

**ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ**

ИИ – искусственный интеллект.

Чат-бот – виртуальный собеседник.

API (Application Programming Interface) – совокупность инструментов и функций в виде интерфейса для создания новых приложений, благодаря которому одна программа будет взаимодействовать с другой.

Токен – средство идентификации пользователя или отдельного сеанса работы в компьютерных сетях и приложениях.

БД – база данных.

ПО – программное обеспечение.

Сервер – программный компонент, выполняющий сервисные функции по запросу клиента, предоставляя ему доступ к определенным ресурсам.

СУБД – система управления базами данных.

Python – высокоуровневый язык программирования общего назначения с динамической строгой типизацией и автоматическим управлением памятью, ориентированный на повышение производительности, является полностью объектно-ориентированным, синтаксис ядра языка минималистичен.

HelpDesk – это программное обеспечение, которое позволяет автоматизировать работу специалистов техподдержки на предприятии. Системы обеспечивают возможность регистрировать заявки и обращения пользователей, например, по вопросам неисправности компьютерной техники, проблемам с программным обеспечением, корпоративной сетью или ИТ-сервисами.

Датасет – набор данных, коллекция из логических записей, хранящихся в виде кортежа.

Telegram – мессенджер, кроссплатформенная система мгновенного обмена сообщениями.

**СОДЕРЖАНИЕ**

[Введение 6](#_Toc106736670)

[1 Описание предметной области 8](#_Toc106736671)

[1.1 Нейронные сети 8](#_Toc106736672)

[1.2 Основные задачи искусственного интеллекта 8](#_Toc106736673)

[1.2.1 Работа с языками. 9](#_Toc106736674)

[1.2.2 Компьютерное зрение 10](#_Toc106736675)

[1.2.3 Распознавание и синтез речи 10](#_Toc106736676)

[1.3 Искусственный интеллект в работе клиентской поддержки 10](#_Toc106736677)

[1.4 Искусственный интеллект в работе технической поддержки 13](#_Toc106736678)

[1.5 Результаты работы аналогичных программ 16](#_Toc106736679)

[2 Теоретический анализ задачи 23](#_Toc106736680)

[2.1 Определение цели и задач 23](#_Toc106736681)

[2.2 Структура проекта 23](#_Toc106736682)

[2.3 Обзор методологий моделирования бизнес-процессов 25](#_Toc106736683)

[2.4 Общие сведения о языке моделирования UML 27](#_Toc106736684)

[2.5 Диаграмма вариантов использования 27](#_Toc106736685)

[2.6 Диаграмма деятельности 29](#_Toc106736686)

[2.7 Диаграмма последовательности 32](#_Toc106736687)

[2.8 Подготовка текстовых данных для подачи в НС 35](#_Toc106736688)

[2.8.1 Bag Of Words 35](#_Toc106736689)

[2.8.2 Embedding 36](#_Toc106736690)

[2.9 Выводу по разделу 37](#_Toc106736691)

[3 Практическая реализация 38](#_Toc106736692)

[3.1 Выбор среды разработки 38](#_Toc106736693)

[3.2 Практическая реализация чат-бота 38](#_Toc106736694)

[3.3 Оценка экономической эффективности 68](#_Toc106736695)

[3.4 Результаты и выводы 71](#_Toc106736696)

[Заключение 72](#_Toc106736697)

[Список использованных источников 73](#_Toc106736698)

[Приложение А Код программы 75](#_Toc106736699)

# **ВВЕДЕНИЕ**

Искусственный интеллект, глубокое обучение и нейронные сети - это термины, которые используются для описания мощных методов, основанных на машинном обучении, они способны решать многие существующие проблемы.

В то время как дедуктивные рассуждения, умозаключения и принятие решений, сравнимые с человеческим мозгом, еще далеки, в последнее время было много достижений в области методов искусственного интеллекта и связанных с ними алгоритмов. Особенно в связи с растущей доступностью больших массивов данных, на основе которых ИИ может учиться [1].

Область искусственного интеллекта опирается на многие области, включая математику, статистику, психологию, физику, машинное обучение, теорию вероятностей, информатику, обработку сигналов, неврологию и лингвистику. Вопросы, связанные с социальной ответственностью и этикой ИИ, проводят параллели со многими разделами философии.

Мотивация для дальнейшего развития методов искусственного интеллекта заключается в том, что решения, необходимые для решения проблем со многими переменными, невероятно сложны, их трудно понять и нелегко собрать вручную [2].

Все чаще исследователи, частные лица и корпорации полагаются на машинное обучение для решения проблем, не требуя подробных инструкций по программированию. Это подход «черного ящика» к решению проблем имеет решающее значение. Программисты – люди находят все более сложным и трудоемким написание алгоритмов, необходимых для моделирования и решения проблем с большим объемом данных. Даже когда мы создаем полезную процедуру для обработки больших наборов данных, она, как правило, чрезвычайно сложна, сложна в обслуживании и невозможна для адекватного тестирования.

Современные алгоритмы машинного обучения и искусственного интеллекта, наряду с хорошо продуманными и подготовленными данными для обучения, способны выполнять программирование за нас.

Интеллект: способность воспринимать информацию и сохранять ее в виде знаний, которые будут применяться для адаптивного поведения в окружающей среде или контексте.

Это определение интеллекта в Википедии может быть применимо как к органическому мозгу, так и к машинному. Интеллект не подразумевает сознания – распространенное заблуждение, распространяемое писателями-фантастами.

Обработка языка и распознавание речи были первыми коммерческими приложениями машинного обучения. Следом появились другие задачи автоматического распознавания (лица, аудио, текст, изображение, шаблон, видео). Сфера применения приложений стремительно расширяется и включает автономные транспортные средства, маркетинг, фильтрацию спама, медицинские диагнозы, компьютерное зрение, игры, поисковые системы, борьбу с преступностью, робототехнику, транспорт, распознавание музыки, классификацию и прочее [3].

ИИ давно укоренился в используемых нами технологиях, сейчас многие не воспринимают его как «ИИ», а принимают за расширение вычислительной техники. Задайте вопрос своему знакомому или просто человеку на улице, есть ли у них искусственный интеллект на телефоне, и вероятней всего, они ответят нет. Но алгоритмы искусственного интеллекта уже давно повсюду, от подсказывающего текста в клавиатуре до системы автофокусировки в камере. Общее мнение таково, что искусственный интеллект еще не появился. Но он давно здесь и существует уже продолжительно время.

ИИ – это довольно обобщенный термин. В центре внимания большинства исследований находится несколько более узкая область искусственных нейронных сетей и глубокого обучения.

# **Описание предметной области**

Рассмотрим наиболее активно развиваемый подход ИИ, его основные задачи.

## **Нейронные сети**

Это направление стабильно держится на первом месте. Продолжается совершенствование алгоритмов обучения и классификации в масштабе реального времени, обработки естественных языков, распознавания изображений, речи, сигналов, а также создание моделей интеллектуального интерфейса, подстраивающегося под пользователя. Среди основных прикладных задач, решаемых с помощью нейронных сетей, финансовое прогнозирование, раскопка данных, диагностика систем, контроль над деятельностью сетей, шифрование данных. В последние годы идет усиленный поиск эффективных методов синхронизации работы нейронных сетей на параллельных устройствах [4].

## **Основные задачи искусственного интеллекта**

Ранее уже было указано, что нельзя дать исчерпывающее определение ИИ. Однако можно перечислить те задачи, методы решения которых на ЭВМ принято связывать с понятием ИИ. Ниже приводятся краткие характеристики таких задач.

Автоматическое решение задач представляет собой не столько вычислительную процедуру поиска ответа, как, например, расчет квадратного корня, сколько нахождение метода решения поставленной задачи. Системы, осуществляющие построение вычислительной процедуры, называют автоматическими решателями задач.

Под распознавателями подразумевают устройства, реагирующие на внешнюю среду через различные датчики, например, видеокамеры, и позволяющие решать задачи распознавания образов. В таких устройствах результаты распознавания выводятся на экран и используются для принятия решений [5].

Например, на современных автоматических боулинговых дорожках установлены видеокамеры, которые после броска шара распознают число и взаимное расположение оставшихся кегель, для подсчета очков. Системы распознавания речи позволяют упростить взаимодействие с компьютером, с помощью упрощенного естественного языка. Необходимо заметить, что существующие на данный момент системы могут распознавать лишь очень ограниченный набор слов-команд, требуют предварительной настройки на дикцию пользователя и не могут анализировать длительную речь (целые предложения), хотя и ведутся интенсивные исследования в этом направлении.

Задачи доказательства теорем и обучения (например, для овладения навыками в какой-либо игре) решаются с помощью автоматического совершенствования алгоритма посредством обработки пробных вариантов с помощью накопления собственного опыта. Следует отметить, что способность к обучению представляет собой одно из основных свойств ИИ.

### **Работа с языками**

Автоматический анализ естественных языков (лексический, морфологический, терминологический, выявление незнакомых слов, распознавание национальных языков, перевод, коррекция ошибок, эффективное использование словарей. Сейчас модели все чаще оцениваются по результатам решения сразу нескольких задач, по их способности к обобщению. Многозадачное обучение приобретает все большее значение, и для него разрабатывают все больше способов оценки качества [6].

### **Компьютерное зрение**

Научное направление в области искусственного интеллекта и связанные с ним технологии получения изображений объектов реального мира, их обработки, использования полученных данных для решения разного рода прикладных задач без участия (полного или частичного) человека. Примеры задач компьютерного зрения: распознавание, идентификация, обнаружение, распознавание текста, восстановление 3D формы по 2D изображениям, оценка движения, восстановление сцены, восстановление изображений, выделение на изображениях структур определенного вида, сегментация изображений, анализ оптического потока.

### **Распознавание и синтез речи**

Распознавание и синтез речи – система решений, позволяющих осуществлять перевод речевого запроса в текстовый вид, в том числе анализ тембра и тональности голоса, распознавание эмоций. Примерами наиболее успешного применения технологии распознавания речи могут служить программные продукты, использующиеся в колл–центрах. Распознавание в составе таких продуктов позволяет в автоматическом режиме обрабатывать простые запросы, контролировать качество оказываемых услуг и оценивать уровень удовлетворённости клиентов [7].

## **Искусственный интеллект в работе клиентской поддержки**

Когда речь заходит о применении разговорного ИИ, первое, что приходит в голову, это клиентская поддержка. По оценкам аналитиков, чат-боты занимают второе место среди самых популярных инструментов цифровизации в российских компаниях (первое место у анализа больших данных и предиктивной аналитики). Их выводы также подтверждают, что чаще всего чат-боты используются для обслуживания клиентов.

Центры обработки вызовов уже давно используют передовые технологии, начиная с систем маршрутизации вызовов, которые направляли потребителей к первым доступным агентам, и заканчивая системами интерактивного голосового ответа, с которыми сегодня взаимодействуют миллионы клиентов [8].

Использование ИИ помогает уменьшать операционные расходы, подбирать персональный подход к обслуживанию клиентов, предоставлять полезную информацию и повышать эффективность агентов – это лишь некоторые из способов, которыми ИИ трансформирует работу организаций по обслуживанию клиентов в настоящее время. Также ИИ освобождает живых агентов от большого стресса и времени.

Центр обработки вызовов является одной из наиболее требовательных сред для искусственного интеллекта, поскольку им необходимо обрабатывать большой объем запросов, соблюдая строгие соглашения об уровне обслуживания. Влияние технологий центра обработки вызовов с искусственным интеллектом на обслуживание клиентов может помочь агентам службы поддержки клиентов работать более продуктивно, вести увлекательные и лично приятные беседы и сократить время, затрачиваемое на простые взаимодействия. Искусственный интеллект может повысить вовлеченность клиентов, повысить лояльность к бренду и повысить удержание.

Искусственный интеллект также может сочетать другие технологии, такие как машинное обучение, глубокое обучение и понимание естественного языка, чтобы преодолеть коммуникационные барьеры и автоматизировать взаимодействие с клиентами. Разговорные чат-боты и рекомендации по продуктам, основанные на данных о поведении клиентов, являются двумя хорошо известными приложениями искусственного интеллекта. Но их возможности применения безграничны.

Маршрутизация на основе навыков появилась в колл–центрах в 1990-х годах – программное обеспечение, которое связывало базовый профиль клиента с агентом, обладающим соответствующими талантами, такими как знание продукта или требуемая помощь. В настоящее время искусственный интеллект расширяет ту же концепцию с помощью прогностической поведенческой маршрутизации. Прогнозирующая поведенческая маршрутизация использует методы и аналитику центра обработки вызовов с искусственным интеллектом для сопоставления абонентов с моделями личности клиентов, которые затем используются для маршрутизации звонков агентам, которые могут наилучшим образом обслуживать этих людей.

Искусственный интеллект может помочь отделам продаж принимать более обоснованные решения для повышения лояльности и удовлетворенности клиентов. Компании все чаще применяют технологии машинного обучения для преобразования сотен типов данных, таких как частота обращения человека за помощью или произнесения фраз «Я закрываю свою учетную запись», в общие показатели потребительского риска. Когда баллы достигают определенного уровня, система отправляет рекомендации по индивидуальным предложениям, такие как скидки или другие льготы [9].

Эмоциональный интеллект – это еще один тип технологии колл–центра с искусственным интеллектом, который может анализировать чувства клиентов во время разговора. Когда клиент раздражен, могут раздаться его голоса, и в разговоре может наступить долгое молчание. Этот тип искусственного интеллекта был обучен различным языкам и культурным условиям, что позволяет использовать его в странах с различными языковыми и культурными традициями. Он использует анализ тона голоса и темпа речи для определения настроения звонящего. Искусственный интеллект также будет оценивать, как часто сотрудник прерывает клиента, и тон голоса как клиента, так и представителя службы поддержки. Затем он предоставит сотруднику живую обратную связь (через всплывающие сообщения), чтобы получить представление о том, что чувствует клиент во время разговора.

Технологии центра обработки вызовов с искусственным интеллектом используются для предоставления полной статистики о времени вызова, разрешении проблемы и другой информации. Инструменты на базе искусственного интеллекта могут выявлять тенденции и получать доступ к данным о клиентах, которые могут помочь менеджерам оценить, они получили положительный или отрицательный опыт. Искусственный интеллект может дать более всестороннюю аналитику, чем менеджер по поддержке клиентов, поскольку он измеряет настроения потребителей, тон и индивидуальность [10].

Клиентская поддержка – это не только про колл-центр, в данный список также попадает и техническая поддержка, которая тоже нуждается в цифровизации.

## **Искусственный интеллект в работе технической поддержки**

Запросы в техподдержку часто не передают полной информации. Сотрудники пишут пару слов: «Не работает» и прикладывают скриншот. При составлении заявки люди не думают о том, что нужно донести суть проблемы, а о том, чтобы быстро сообщить о ней и вернуться к своей работе, что абсолютно логично. Сотрудникам первой линии поддержки приходится им перезванивать или писать в личные чаты, чтобы уточнить информацию.

Благодаря инструкциям и ответам бота ИИ будут решаться стандартные вопросы на нулевой линии поддержки и это облегчит работу сотрудников второй линии. Некоторые пользователи ввиду отсутствия знаний по исправлению проблемы или ошибки делают заявку в техподдержку и тратят время на ожидание ответа от инженера, который может быть занят другими задачами. Большую часть проблем пользователь, может решить самостоятельно изучив инструкцию по своей проблеме, тем самым освободив сотрудника от простых задач, что позволит освободить время для сложных задач.

Раньше задача по обработке запросов выполнялась сотрудниками. При большом количестве запросов, распределение команд и специфика проблем она имеет рутинный характер. Убрав ручной труд, увеличивается скорость и точность выполнения заявок. Иногда сотрудник, распределяющий заявки, может не знать о характере той или иной проблемы, и кто должен быть исполнителем в этой заявке. Например, сотрудник по незнанию кинул заявку на сотрудника, находящегося в отпуске, и данная заявка будет отработана не своевременно, что вызовет просрочку. Также бот сокращает общее время жизни заявок за счет быстрой передачи на нужного сотрудника, что даст плюс в SLA по времени взятия заявки в работу. Автоматическое распределение заявок может происходить в зависимости от уровня квалификации специалиста. Пример организации службы поддержки в компании приведён на рисунке 1.1 [11].



Рисунок 1.1 – Пример организации службы поддержки

Одна из важных функциональных возможностей – распознавание изображения. Когда приходит заявка с изображением, модуль может распознавать на нем особенности информационных систем (логотип), стандартные для системы диалоговые окна и текст используемый в них. По этим критериям модуль определяет, какая система изображена на скриншоте и, соответственно, понимает в какую группу, ответственную за эту систему, отправить данную заявку. Для того, чтобы определение было точнее модуль распознает текст в диалоговом окне, которое сообщает об ошибке.

Внедрение распознавания речи. Когда сотрудник звонит в техподдержку, но нет свободных операторов, то IVR предлагает продиктовать проблему и задаёт нужные для грамотного составления заявки вопросы, типа «какая у вас система», «в чем заключается проблема». После звонка создаётся заявка, бот приложит файл с записью разговора, а она будет переведена в текст, и этот текст в свою очередь, пройдет через лингвистический модуль. За счёт этого автоматизируется обработка заявок, поступающих в ServiceDesk.

Модуль нужно обучения на данных, для полного включения в работу. В компании присутствуют старые и важные данные – это старые заявки, обращения, написанные инструкции и известные результаты: кто исполнитель, какое решение принято. Модуль, в условиях корректного описания заявки и пользуясь заложенной в него логикой, может разобрать эту информацию, увидеть зависимости между определенными словами в тексте и назначенными сервисами. Затем бот обрабатывает заявки таким образом: распределение по условным исполнителям и прикрепление данных по предыдущим запросам или уже созданным инструкциям, если они имеют схожую проблематику, чтобы сотрудник мог посмотреть, как заявка была решена ранее. Данная логика также позволит сократить время решения

Система никак не заменит операторов, но облегчает им работу, избавляя от большого количества повторяющихся запросов. Для сотрудников освобождается время для тщательного рассмотрения других, более важных и сложных обращений.

Большое количество информации, находится в различных документах и у разных операторов, она разбросанная по системам, постепенно собираются в объединённую базу знаний. Ответы на вопросы и поиски решений проблем становится существенно проще.

Плюсы использования чат-ботов в клиентской поддержке.

Более быстрое решение вопросов. Сотрудник не способен общаться с несколькими клиентами одновременно. Боты же не имеют подобных ограничений. Их легко можно запрограммировать на автоматические ответы десяткам или сотням клиентов одновременно, в зависимости от требований компании. Также боты являются хорошим источником для быстрых и простых ответов на часто задаваемые вопросы. Если учесть, что большинство запросов имеют простые решения, нет смысла нанимать много сотрудников и заставлять клиентов подолгу ждать своей очереди. Боты решают простые проблемы быстро, можно сказать моментально.

Круглосуточная поддержка. Без использования ботов человеческая техподдержка, доступна 24/7 только, если компания составила для сотрудников не только дневные, но и ночные смены. Люди, не способны работать круглосуточно без перерывов и с одинаковой интенсивностью, но это способны делать чат-боты.

Качество сервиса не становится хуже. Чат-боты никогда не испытывают чувства голода, усталости, они не теряют терпения, не теряют работоспособность из-за своего состояния или других внешних факторов, не теряют производительности, и не отличаются между собой, благодаря этому он будет вести себя одинаково с каждым клиентом, что является залогом высокого уровня обслуживания [12].

## **Результаты работы аналогичных программ**

Если не автоматизировать техподдержку крупной компании, то она будет перегружена частыми и простыми обращениями, что повлечёт за собой плохое решение сложных вопросов, в виду недостатка времени и усталости сотрудника.

Чтобы снять нагрузку с технической поддержки, требуется интеллектуальный промежуточный слой. Он позволит решать стандартные вопросы на нулевой линии поддержки с помощью чат-бота и существенно облегчит работу операторов первой линии.

Helpdesk – не проблема, в наше время на рынке имеется много готовых решений, но запустить своего чат-бота уже сложнее. Чтобы клиенты не чувствовали дискомфорта и не стали относиться к нему отрицательно, требуется машинное обучение, а это значит: дата-сайентисты, лингвисты и тщательная разметка данных. Следом нужно интегрировать бота с helpdesk и корпоративными сервисами, далее нужно проследить, чтобы работа шла хорошо и не возникало ошибок. С виду AutoFAQ схож со стандартным helpdesk со списком тикетов, разделением по тематикам вопросов, очередями и окном чата. Также из интерфейса доступны дашборды, история переписки, отчетность.

Когда пользователь связывается с техподдержкой по почте, через мессенджер, сайт или одну из соц. сетей, начинают работать алгоритмы машинного обучения. Классификатор определяет, в чем состоит вопрос, к какой базе обратиться, и какая запись в ней содержит наиболее подходящий ответ. Пока сотрудник общается с клиентом, программа анализирует диалог и догадывается, какой ответ или действие подходят в этом случае. Когда бот поймет, что к чему, сотрудник может снова переключить заявку на него и заниматься другой работой.

Если вопрос оказался новым для бота, или в базе нет ответа, который бы подходил к нему, сотрудник разберется с проблемой самостоятельно. Ему доступны такие же системы, с которыми работает программа.

Программа должна дообучиться на данных, которые собираются во время работы. Задача осложнена тем, что обучение должно проходить без помощи лингвистов и data-scientistов. Поэтому решили, что перед запуском не будет данных вовсе, либо будет некорректная выборка. Например, 80% вопросов будет относиться к нескольким проблемам, а для других будет по одному вопросу.

Чтобы программа работала «из коробки», будет использована нейронная сеть, которая была предобучена на различных проектах. Она отвечает за общение с пользователя техподдержки, пока AutoFAQ не соберет данные на практике.

Это позволяет запускать AutoFAQ в работу практически без подготовки и сразу получать достаточно высокое понимание вопросов. Благодаря этому чат-бот понимает вопросы, которые были сформулированы совершенно иначе, чем в примерах, поэтому он абсолютно нетребователен к объему и качеству выборки для обучения.

У этого подхода есть недостатки. Чат-бот хорошо понимает общую лексику, но специфические выражения поначалу даются ему плохо. Поэтому в первое время он будет часто ошибаться, и нагрузка на техподдержку будет уменьшаться постепенно. Сразу после запуска программа начинает накапливать обучающую выборку и повышать качество понимания вопросов. Чтобы она быстрее начала понимать специфические термины, предусмотрен словарь, который можно заполнить вручную [13].

Бот обучается, если пользователи или сотрудники подтверждают связь вопроса с ответом, хранящийся в базе знаний: либо пользователь в диалоге с ботом дает понять, что ответ верный, либо сотрудник техподдержки кликает по подсказкам, как в примере выше. Благодаря этому накапливается датасет из различных формулировок одного и того же вопроса. Это помогает чат-боту лучше понимать пользователей.

Второй механизм, с помощью которого чат-бот наращивает объем знаний – пополнение баз новыми примерами вопросов и записями. Их можно как составить самостоятельно и загрузить вручную, но чаще используется полуавтоматическое пополнение. Каждый раз, когда система самостоятельно не справляется с обработкой запроса, ей на помощь приходит оператор, а в систему создаются рекомендации для пополнения баз. После помощи оператора, его ответ направляется в реестр рекомендаций. Следом администратор системы просматривает реестр, выбирает и разбивает по группам удачные рекомендации и вносит их в базу знаний. Собранная таким образом выборка получается несбалансированной, но она состоит из вопросов, которые актуальны здесь и сейчас.

Когда AutoFAQ нарабатывает необходимый объем данных, предобученный алгоритм отключается и в дело вступают модели, которые используют обучение с учителем. Нельзя предугадать заранее, какой из методов лучше справится с определённым клиентом, поэтому в AutoFAQ встроены сразу несколько алгоритмов: от простых TF-IDF, до BERT и LASER.

Система проверяет объем и сбалансированность обучающей выборки. И в зависимости от результатов формируется новый программный конвейер, подбираются алгоритмы и гиперпараметры.

Вначале используются простые модели – логистическая регрессия и SVM. По мере накопления данных в дело вступают нейронные сети. Выбор алгоритма происходит на основе подсчета метрик классификации или при помощи предустановленных требований к обучающей выборке. На практике, в системе часто строятся ансамбли из нескольких алгоритмов. Она постоянно сравнивает результаты по метрикам и выдает на выход наиболее лучшие ответы.

Простой интерфейс базы знаний позволяет без каких-либо знаний создавать новые классы, понимать причины ответов и изменять их, внося изменения. Так что, программой легко управлять, даже когда собирается большая обучающая выборка.

Система не требует подключения к сторонним сервисам и работает в изолированной среде. Все ее компоненты дублируются. Они либо работают параллельно, либо запускаются по мере надобности.

На примере программы AutoFAQ, показанной на рисунке 1.2, от компании КРОК можно наблюдать, как улучшается автоматизация чат-бота.

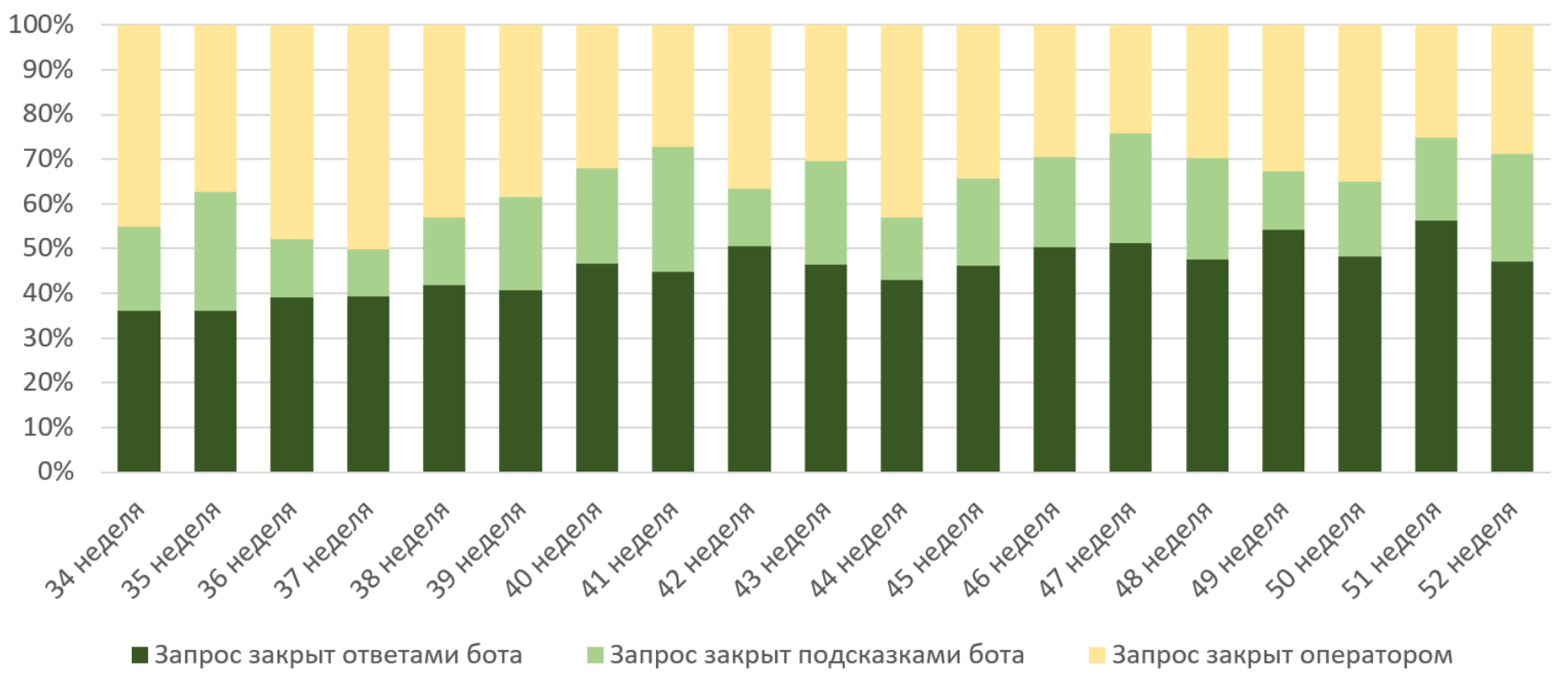


Рисунок 1.2 – Результаты работы AutoFAQ

AutoFAQ разворачивается за несколько часов, подключается ко многим текстовым каналам коммуникации от электронной почты до Skype for business. После этого система готова к работе – она понимает письменную речь, начинает обрабатывать заявки в техподдержку и записывать все это в собственную базу знаний. Чем больше заявок будет поступать, тем лучше бот научится их понимать и тем больше обращений он будет брать на себя. За несколько месяцев он наполняет свою базу и автоматизация запускается полностью. В большинстве случаев, AutoFAQ будет решать около 45% заявок самостоятельно.

Рассмотрим чат-бота «Лию» от компании Lia Platform.

Раньше команда делала чат-боты для бизнеса: создали Facebook-бота для «Точки», ассистентов для «Рокетбанка», Simple Wine и еще десятка проектов. Но в 2016 году компания решила сделать ботов более умными и автоматизировать поддержку нейронными сетями [14].

Первым заказчиком Lia стал банк «ВТБ» – им нужно было автоматизировать поддержку в новом интернет-банке для предпринимателей. Дедлайн заставил в сжатые сроки доработать технологию: нейронные сети стали заворачиваться в серьезный IT-продукт. С «ВТБ» компания запустилась достаточно успешно – особенно для продукта, который был сделан за четыре месяца.

Бот базировался на нейросетях и сразу был умным: отвечал больше чем на 800 вопросов, поддерживал несколько сложных сценариев (выписки, смена тарифа, настройка пользователей) и говорил, как человек.

Следующим кейсом стала компания «Ситимобил», которая занимается предоставлением услуг такси.

Задача – обеспечить бесперебойную работу сервиса и повысить лояльность пользователей за счет автоматизации службы поддержки в приложении.

Действия Лии – научиться закрывать не менее 15% кейсов самостоятельно. Бот встал на первую линию и забрал на себя высокочастотные запросы, требующие оперативной реакции: отмену заказа, определение бонусной программы пользователя по уровню лояльности. В результате Лия превзошла все ожидания и достигла 69% уровня автоматизации – это в 4,5 раза больше, чем рассчитывали изначально.

Результат – скорость обработки запросов пользователей выросла в десять раз, а Лия превзошла KPI в 4,5 раза и покрывает 69% обращений в службу поддержки.

Затем в услугах Лии заинтересовалась компания «Самокат», занимающаяся доставкой продуктов.

Задача – сократить время ответа оператора на популярные запросы пользователей – 19% от общего количества обращений.

Действия Лии – встать на первую линию и взять на себя наиболее популярные вопросы: узнать статус заказа, время доставки и уточнить, как работает приложение. Также был реализован функционал поддержки операторов: сотрудники получают обращения сразу со всей необходимой информацией и быстро решают проблему пользователя без лишних уточнений.

Результат – Лия достигла 51,2% покрытия всего лишь за месяц, а сейчас обрабатывает 61% запросов без участия человека. Скорость и качество клиентской поддержки увеличились в несколько раз.

Новый кейс – компания «Whoosh», сервис проката самокатов.

Задача – оптимизировать расходы на увеличение штата и сократить стоимость обработки запроса без потери качества.

Действия Лии – задача закрыть большой поток высокочастотных однотипных запросов пользователей: до этого компания планировала увеличить штат службы поддержки в 2 раза из-за большого количества обращений. Компания реализовала сценарии, в которых Лия обрабатывает заявки на добавление новой / расширение зоны парковки, предоставляет пользователям информацию о подписке, уточняет правила парковки и завершения аренды.

Результат – компания отказалась от найма 90 человек в службу поддержки: Лия самостоятельно отвечает на 41% запросов, а обработка одного сложного обращения стоит дешевле минимум в 3 раза.

Такой помощник может разговаривать с человеком по телефону или в чате. Он сразу извлекает кучу информации из естественной речи: e-mail, запросы, адреса. На чат-бота можно скинуть большинство рутинных задач: от доставки пиццы и вызова такси до консультации клиентов. Сборка умного ассистента в платформе занимает от двух недель до месяца. В первые дни технология начинает закрывать до 30% обращений, а через полгода — до 70%.

За шесть месяцев робот учится реалистично понимать запросы и узнавать паттерны: все это время его курирует живой человек [15].

# **Теоретический анализ задачи**

* 1. **Определение цели и задач**

Цель работы: разработка прототипа чат-бота для классификации обращений граждан на основе нейросетей.

Объект исследования: обращения граждан.

Предмет исследования: классификация категории по смыслу обращения.

Задачи работы:

* анализ предметной области;
* моделирование освоенных бизнес-процессов;
* выбор среды разработки;
* выбор способа обработки текста;
* выбор модели обучения для нейросети;
* поиск необходимой информации для подготовки датасета;
* разработка и отладка нейросети и чат-бота.

Проблема сервиса на данном этапе: отсутствие данного функционала во многих компаниях, частое выполнение однотипной работы, желание автоматизировать работу сотрудников.

* 1. **Структура проекта**

Разработка ИТ-проекта по разработке классификатора включает в себя пять этапов, три из которых основных:

* подготовка проекта;
* анализ;
* разработка и тестирование.

Данный проект содержит по нескольку уровней декомпозиции в каждом этапе, всего в проекте двадцать одна работа.

*Первый этап.* Начало работ.

*Второй этап.* Подготовка проекта:

* предварительное планирование проекта;
* определение ожидаемых результатов и сроков проекта.

*Третий этап.* Анализ:

* анализ бизнес-процессов управления клиентского сервиса;
* построение диаграммы вариантов использования as is;
* построение диаграммы деятельности as is;
* построение диаграммы последовательности as is;
* построение диаграммы вариантов использования to be;
* построение диаграммы деятельности to be;
* построение диаграммы последовательности to be;
* анализ технических и программных средств реализации проекта.

*Четвертый этап.* Разработка и тестирование:

* разработка алгоритма работы чат-бота;
* установка необходимых библиотек для Python;
* создание датасета;
* наполнение датасета;
* написание кода программы на языке Python;
* интеграция с мессенджером «Telegram»;
* реализация разработанных макетов интерфейса;
* проведение функционального тестирования;
* устранение выявленных несоответствий.

*Пятый этап.* Завершение работ.

Все этапы проекта следуют друг за другом, и большинство работ выполняются последовательно. Неудачи и успехи фиксируются для более успешной работы в дальнейшем.

## **Обзор методологий моделирования бизнес-процессов**

Главной целью использования методологий и методов моделирования бизнес-процессов является повышение операционной эффективности компании – то есть организация всех дел наиболее оптимальным способом, ведущим к снижению затрат и одновременно к улучшению качества предлагаемых продуктов или услуг. Для того, чтобы провести такого рода оптимизацию, нужно в первую очередь смоделировать основные процессы, повседневно происходящие во всех подразделениях предприятия.

К настоящему времени разработаны многочисленные методологии моделирования бизнес-процессов. К выбору стоит подходить ответственно – в конечном итоге именно от этого зависит, достаточно ли наглядной, удобной и понятной окажется модель. Поэтому конкурентоспособный управленец обязан быть хорошо знаком как минимум с несколькими основными методиками.

Flow Chart Diagram (диаграмма потока работ) – способ графического описания работы с применением особых символов для каждой операции, набора данных, единицы оборудования, исполнителя. В результате на схеме демонстрируется логическая последовательность всех операций. Это гибкий подход, он дает возможность при необходимости рассмотреть один комплекс действий сразу в нескольких вариантах.

Data Flow Diagram – изображение передачи данных между операциями, для характеристики информационной стороны бизнес-процесса. Это позволяет наблюдать данные на входе в систему и в каждую операцию в отдельности, и соответствующую информацию на выходе. Также в ней отображается, какими способами сведения претерпевают изменения и где хранятся. Деятельность компании раскладывается на логические информационные уровни, причем базовая схема улучшается добавлением подробных описаний подпроцессов, имеющих свою внутреннюю структуру.

IDEF (Integrated Definition for Function Modeling) – это целый набор аналитических средств, применяемых не только в управлении бизнесом, но и во многих других сферах. Чаще всего встречаются варианты IDEF0 и IDEF3. Первый из этих вариантов представляет собой модель функций, причем сложные функции делятся на более простые составляющие, а затем различные блоки логически объединяются посредством стрелок. При использовании IDEF3 речь идет о «поведенческом» описании: демонстрируется поток работ либо переходные состояния объектов.

Unified Modeling Language – графический язык для визуализации, специфицирования, конструирования и документирования процессов и систем. Комплекс из девяти видов диаграмм, описывающих разные аспекты: классы, объекты, прецеденты, последовательности, кооперации, состояния, деятельность, компоненты, развертывание. В результате получается представление очередности действий сотрудников и работы различных объектов внутри организации. Схема может разветвляться, в ней отмечаются разнообразные условия и исключения из правил.

ARIS (Architecture of Integrated information Systems) – методология и соответствующее семейство программных продуктов. Они используются для структурированного описания, анализа и последующего совершенствования бизнес-процессов предприятия. Система наглядно показывает правила деятельности предприятия и значения показателей результативности. Так можно определить желаемые характеристики работы компании, совершенствовать архитектуру, улучшить процессы, рационально распределять ресурсы. Инструмент определяет весь цикл разработки – анализ требований, спецификация информационной системы и описание физической реализации.

Выбор методологий должен базироваться на четком понимании их возможностей и недостатков, а также целей использования создаваемых моделей бизнес-процессов [16].

## **Общие сведения о языке моделирования UML**

UML (Unified Modeling Language – унифицированный язык моделирования) – язык графического описания для объектного моделирования в области разработки ПО, а также для моделирования бизнес-процессов.

В UML выделяют девять типов диаграмм:

* диаграммы классов (Class diagram);
* диаграммы объектов (Object diagram);
* диаграммы вариантов использования (Usecase diagram);
* диаграммы последовательностей (Sequence diagram);
* диаграммы кооперации (Collaboration diagram);
* диаграммы состояний (State diagram);
* диаграммы деятельности (Activity diagram);
* диаграммы компонентов (Component diagram);
* диаграммы развертывания (Deployment diagram).

Процессы представлены в нотации UML, так как данная нотация позволяет показать работу системы с разных сторон. А также для представления модели базы данных будет использоваться ER-диаграмма [17].

## **Диаграмма вариантов использования**

Диаграмма вариантов использования (сценариев поведения, прецедентов) является исходным концептуальным представлением системы в процессе ее проектирования и разработки. Данная диаграмма состоит из актеров, вариантов использования и отношения между ними.

Такие диаграммы помогают выявить:

* сценарии, в которых система или приложение взаимодействуют с людьми, организациями или внешними системами;
* цели, которые система или приложение помогает этим актерам достичь;
* сферы применения системы.

Диаграмма вариантов использования включает в себя следующие компоненты: актер, варианты использования, ассоциации.

Актеры: пользователи, которые взаимодействуют с системой. Актером может быть человек, организация или внешняя система, которая взаимодействует с приложением или системой. Это должны быть внешние объекты, которые производят или потребляют данные.

Варианты использования: горизонтальные овалы, которые представляют различные варианты использования, которые может иметь пользователь.

Ассоциации: связь между актером и вариантами использования.

На рисунке 2.1 представлена диаграмма вариантов использования «as is».

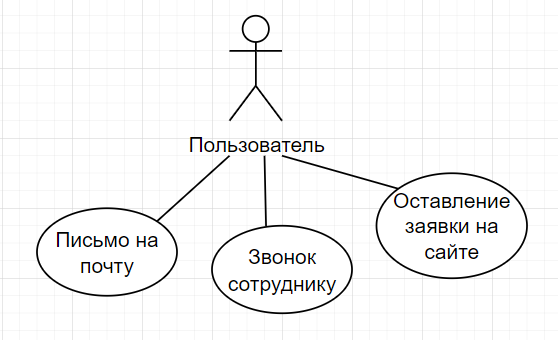


Рисунок 2.1 – Диаграмма вариантов использования «as is»

С внедрением чат-бота у пользователя останется лишь несколько функций, подача заявки и изучение полученной инструкции для самостоятельного решения проблемы. Диаграмма вариантов использования «as to be» предоставлена на рисунке 2.2.

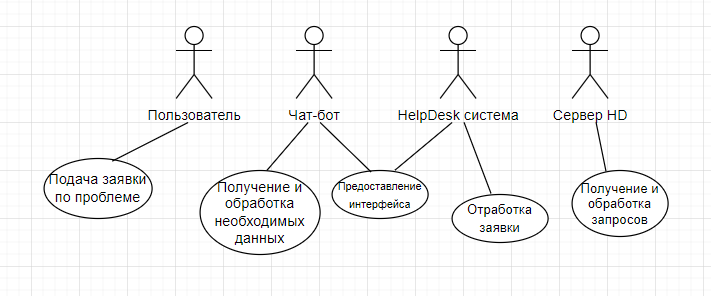


Рисунок 2.2 – Диаграмма вариантов использования «as to be»

## **Диаграмма деятельности**

При моделировании поведения проектируемой или анализируемой программной системы возникает необходимость не только представить процесс изменения ее состояний, но и детализировать особенности алгоритмической и процедурной реализации выполняемых системой операций.

Для этой цели, как правило, используются блок-схемы или структурные схемы алгоритмов. Каждая такая схема акцентирует внимание на последовательности выполнения определенных процедур или элементарных операций, которые в совокупности приводят к получению желаемого результата. Для моделирования процесса выполнения операций в языке UML используются диаграммы деятельности.

В рамках данного типа диаграмм будет рассматриваться один из основных бизнес-процессов данной системы – направление обращения о проблеме пользователем через систему. На рисунке 2.3 показан процесс отработки обращений с точки зрения работы сотрудников «as is».

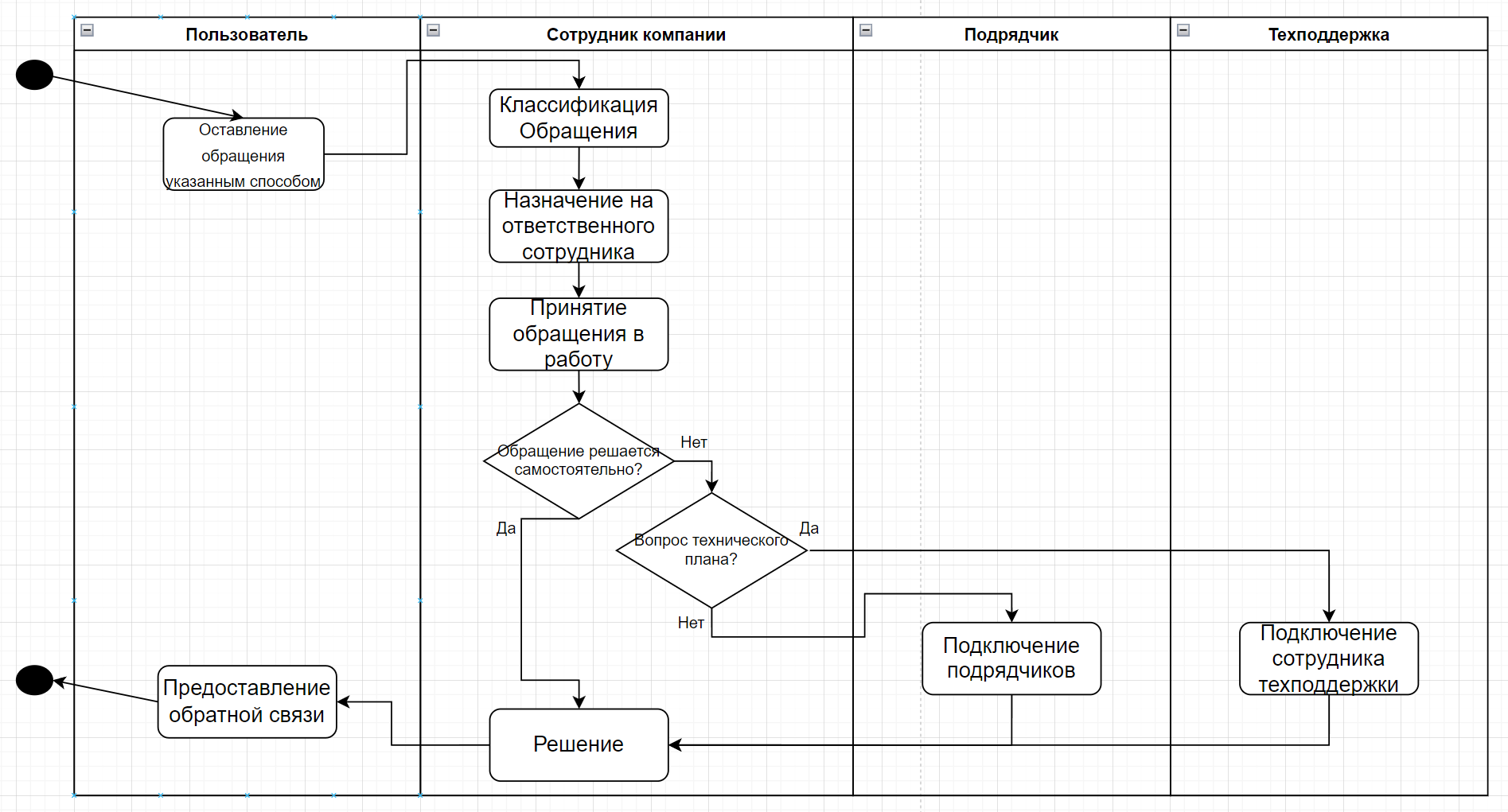


Рисунок 2.3 – Диаграмма деятельности «as is»

На рисунке 2.4 показан процесс отработки обращений с точки зрения работы программы «as is».

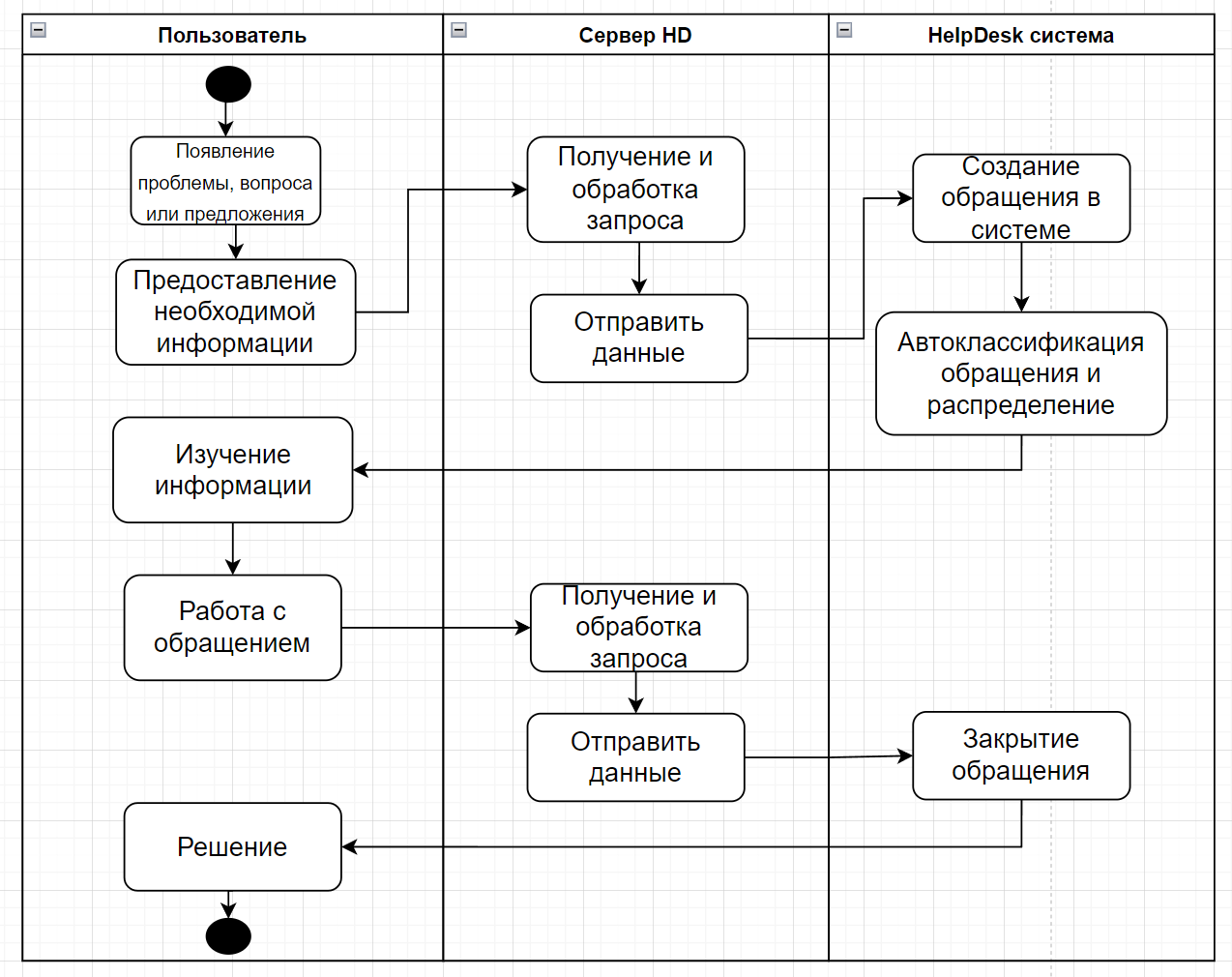


Рисунок 2.4 – Диаграмма деятельности «as is»

С внедрением чат-бота сотруднику не нужно будет распределять данные обращения на ответственные подразделения и на конкретного сотрудника, это все будет делать система.

Недостатки, обнаруженные в модели «as is», исправляются путем создания модели «as to be» (как это будет), то есть моделей организации новых процессов. Использование функциональной модели «as to be» позволяет не только сократить время внедрения ИТ-системы, но и снизить риски, связанные с сопротивлением персонала информационным технологиям. Модель to be необходима для анализа альтернативных (наилучших) способов выполнения функций и документирования работы компании в будущем. Функциональная модель «as to be» четко определит распределение ресурсов между операциями бизнес-процессов, что позволит оценить эффективность использования ресурсов после предложенного перепроектирования. Итак, модель to be состоит также из четырех этапов:

* обращение клиента;
* идентификация клиента;
* обработка обращения;
* закрытие заявки.

В модели «as to be» заменился один из участников, рассматриваемого бизнес-процесса, таким образом в новой модели список участников такой: Пользователь, чат-бот, Сервер HD, HelpDesk система. Рассмотрим подробнее сам процесс обработки обращений клиентов в чат после внедрения чат-бота. Точкой входа данного процесса является обращение клиента в чат, а точкой выхода – закрытие чата. Таким образом в следствии введения нового программного продукта, а именно чат-бота, все проблемы, выявленные в модели «as is», устранятся и сделают работу сотрудников управления клиентского сервиса удобнее.

На рисунке 2.5 показана диаграмма деятельности с точки зрения работы сотрудников «as to be».

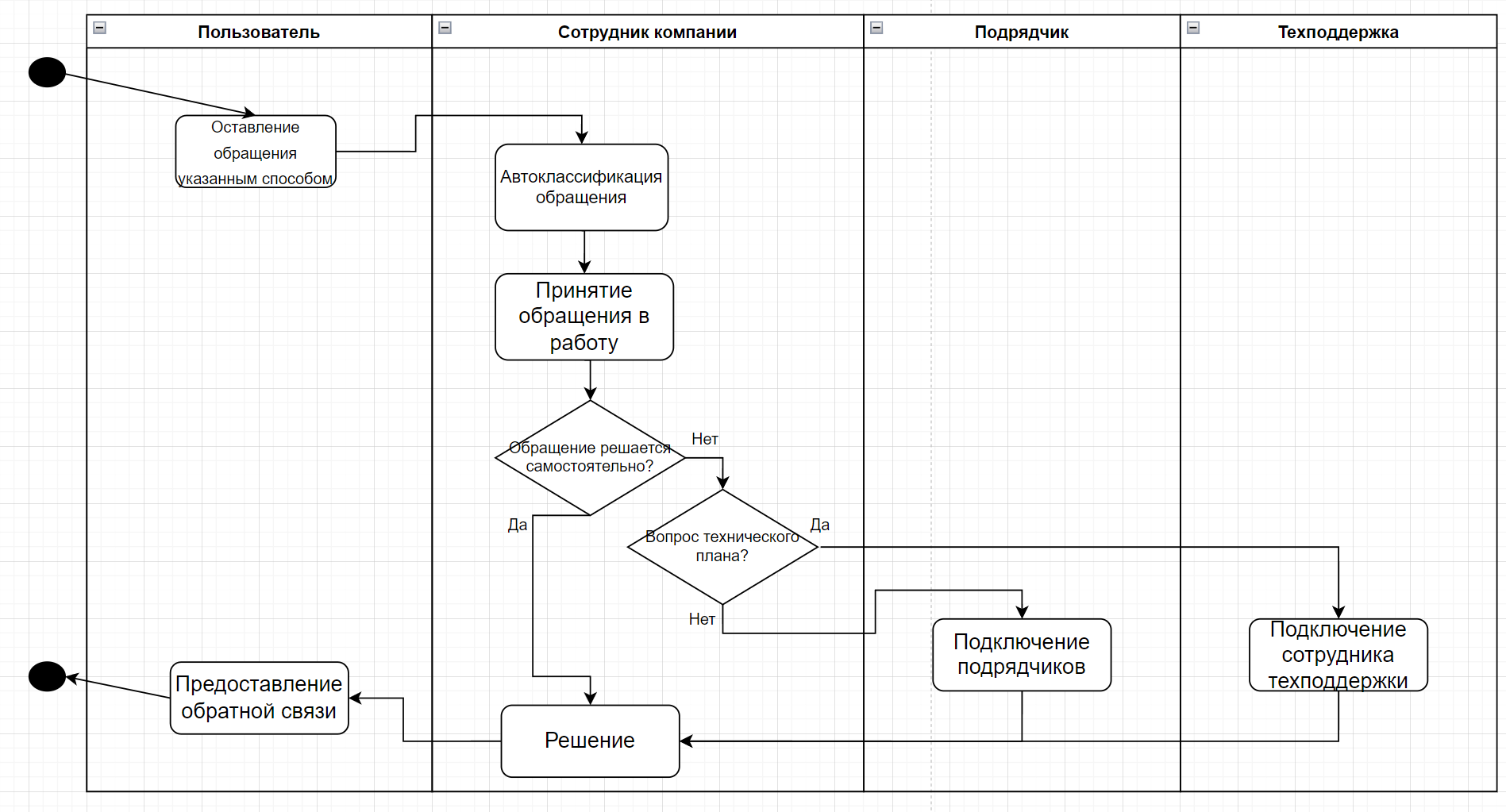


Рисунок 2.5 – Диаграмма деятельности «as to be»

На рисунке 2.6 показан процесс отработки заявок с точки зрения работы программы «as to be».

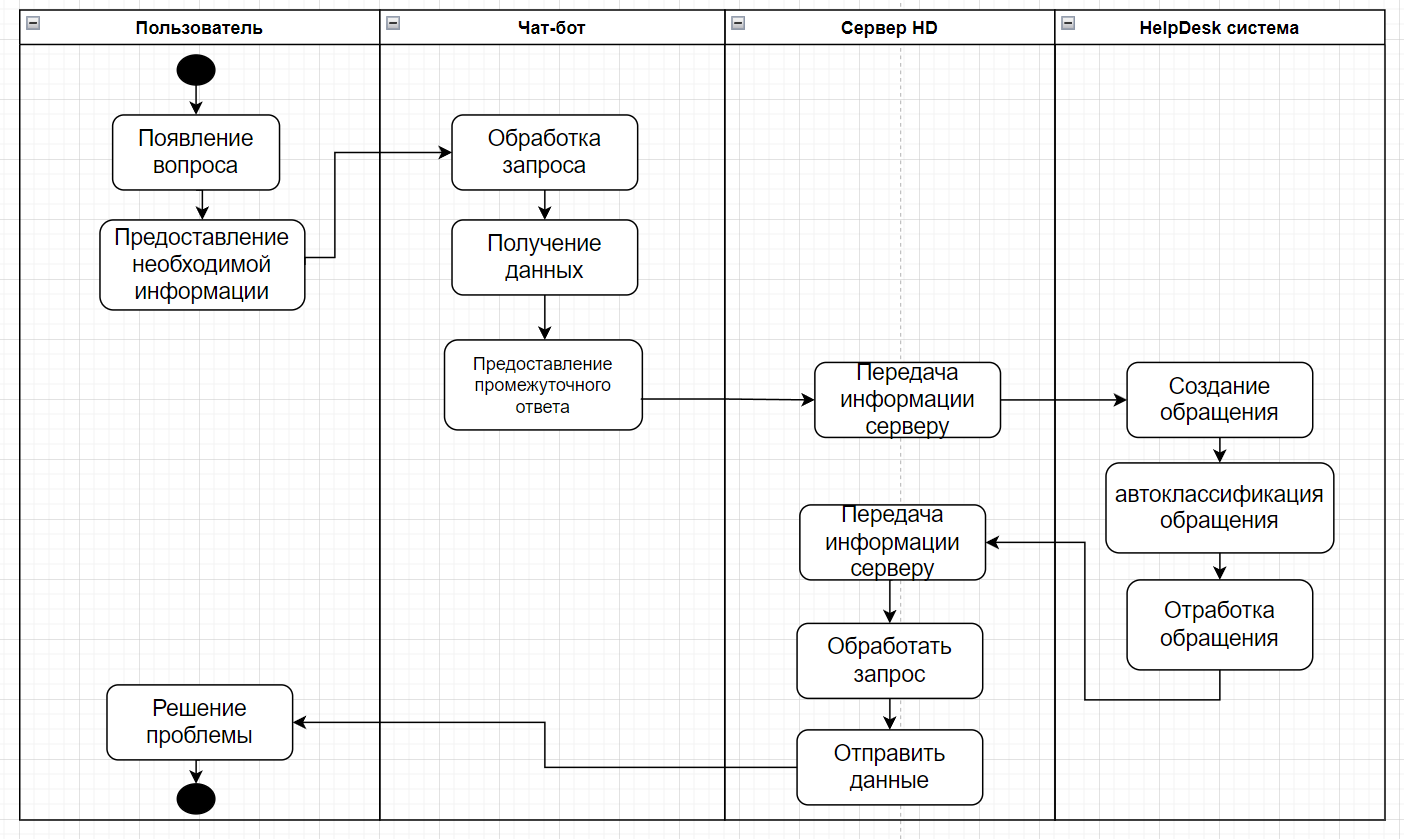


Рисунок 2.6 – Диаграмма деятельности «as to be»

## **Диаграмма последовательности**

Диаграммы последовательности (sequence diagram) являются видом диаграмм взаимодействия языка UML, которые описывают отношения объектов в различных условиях. Условия взаимодействия задаются сценарием, полученным на этапе разработки диаграмм вариантов использования.

Диаграммы последовательности фиксируют:

* взаимодействие, происходящее при совместной работе, которая реализует сценарий использования или операцию;
* взаимодействия высокого уровня между пользователем системы и системой, между системой и другими системами или между подсистемами.

Основными элементами диаграммы последовательности являются обозначения объектов (прямоугольники с названиями объектов), вертикальные «линии жизни», отображающие течение времени, прямоугольники, отражающие деятельность объекта или исполнение им определенной функции (прямоугольники на пунктирной «линии жизни»), и стрелки, показывающие обмен сигналами или сообщениями между объектами.

Линия жизни представляет каждого участника взаимодействия.

Активность представляет период, в течение которого элемент выполняет операцию. Верх прямоугольника обозначает время начала, низ прямоугольника обозначает время завершения.

Сообщение определяет конкретную связь между линиями жизни.

На диаграмме последовательности можно увидеть следующие аспекты:

* сообщения, побуждающие объект к действию;
* действия, которые вызываются сообщениями – передача сообщения следующему объекту или возвращение определенных данных этого объекта;
* последовательность обмена сообщениями между объектами.

На рисунке 2.7 показана диаграмма последовательности «as is».

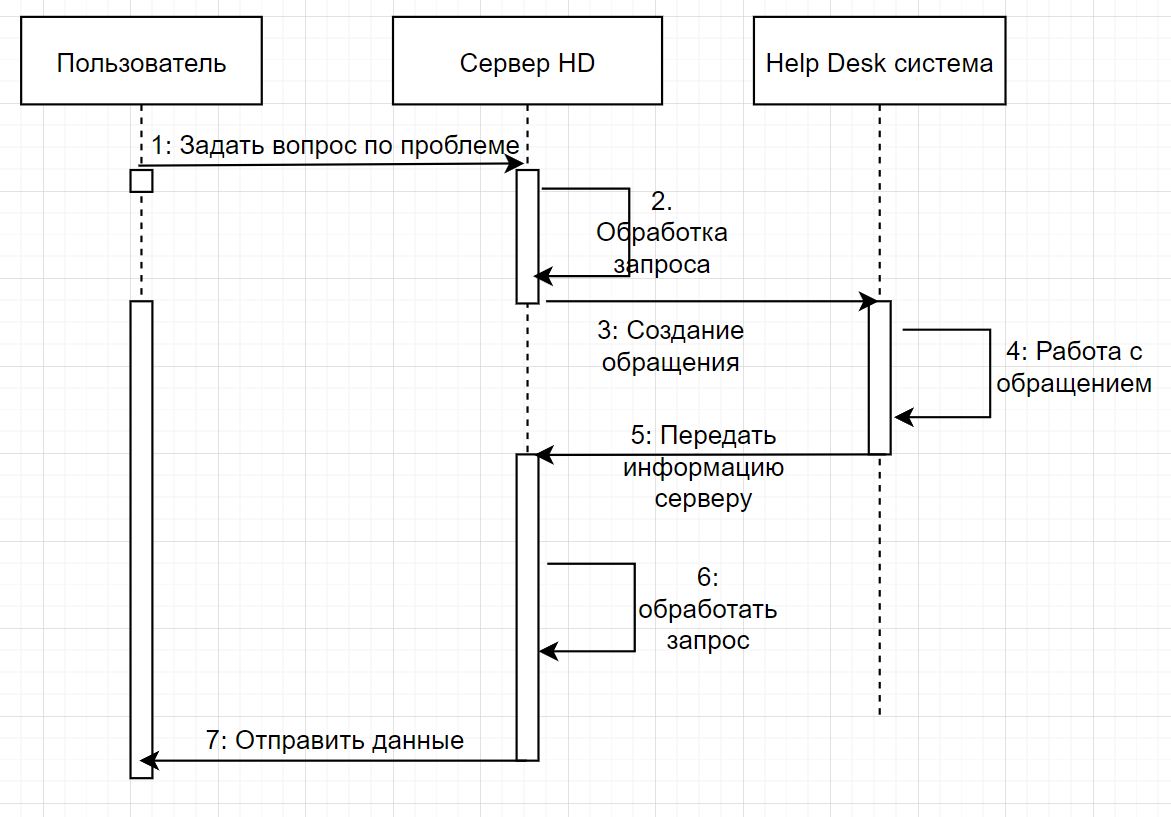


Рисунок 2.7 – Диаграмма последовательности «as is»

С внедрением чат-бота пользователь сможет без подключения сотрудника технической поддержки получить необходимую информацию, как только откроет приложение.

На рисунке 2.8 показана диаграмма последовательности «as to be».

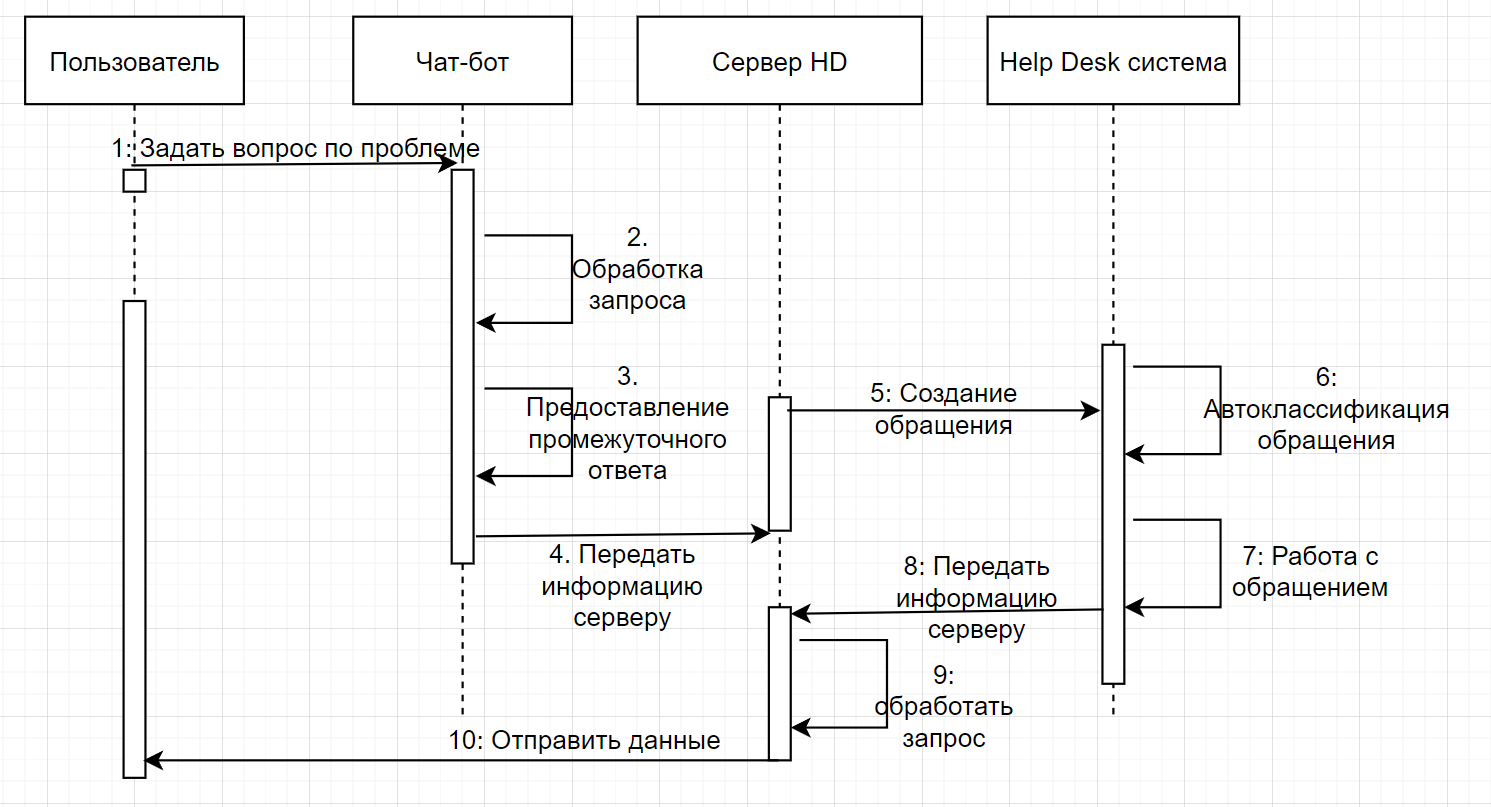


Рисунок 2.8 – Диаграмма последовательности «as to be»

Диаграммы последовательности предназначены для моделирования взаимодействия между несколькими объектами. Зачастую эти диаграммы создаются для моделирования взаимодействия между объектами в рамках одного прецедента. Как видно из диаграмм, задача обработки обращений для сотрудника была существенно облегчена.

* 1. **Подготовка текстовых данных для подачи в НС**

Нейронная сеть работает только с цифровыми данными. Поэтому информацию любого вида (текст, изображение) разработчик должен специальным образом подготовить.

Текстовые данные невозможно напрямую перевести в матрицу коэффициентов, но можно воспользоваться возможностью разделить слова на символы и поставить им в соответствие определенные цифры, например. Этот метод называется токенизацией. Библиотека Keras в Python содержит метод tokenizer(), который принимает на вход текст, а возвращает словарь частотности. Он сам отфильтрует ненужные символы, составит множество пар «слово - количество повторений слова», расположит все полученные значения обратном порядке. Самые частые слова текста (союзы, частицы, предлоги, местоимения) займут место в начале списка индексов, а самые редкие окажутся в конце.

Следующий этап, после подготовки и фильтрации текста – создание выборки для обучения. На текущий момент текст представляет собой одну большую последовательность индексов – один вектор. В обучающей выборке должно содержаться большое количество примеров, чем больше – тем лучше, но для подачи в нейросеть этот вектор необходимо поделить.

### **2.8.1 Bag Of Words**

Это метод порядкового преобразования последовательности индексов в вектор из 0 и 1. Длина вектора задатся вручную, обычно это от 10000 до 30000 значений, ее условились обозначать как MaxWordsCount, однако возможны и произвольные названия.

Суть метода в том, что берется первые 10000 к примеру, пар слов-токен из словаря частотности и проверяем, какие именно токены присутствуют в обрабатываемом фрагменте текста:

* индексы слов, встретившихся в диапазоне до 10000 превратятся в единицы, но в соответствии с их порядковым местом с начала отсчёта: индекс союза;
* все индексы, которые пропущены превратятся в нули.

Выполнение: создается вектор длиной в max\_words и заполняется нулями. Далее разбивается исследуемый текст на куски и анализируется, встречается ли лексика из словаря частотности, и если да, то какая именно.

### **2.8.2 Embedding**

По смыслу – умная нормализация. Каждый токен преобразуется в вектор вещественных чисел. И это делается не случайным назначением, а обучением. Embedding-слой создаёт скрытое пространство, внутри которого для каждого индекса слова будет создан свой отдельный вектор с новыми значениями.

Вектор многомерный, это точка в n-мерном пространстве. Векторы к токену должны быть логичными, т.е. синонимы располагаются в одном пространственном облаке, а антонимы наоборот.

Если вектора располагать хаотичными, дольше обучение и ниже сходимость, также теряется возможность понимания смысла текста машиной. Потому что по пространственной принадлежности она имеет установки – здесь хорошее, здесь плохое.

Очевидное преимущество слоя Embedding в том, что при его использовании группы слов текста получают новые координаты, благодаря чему индексы слов объединяются в области по их принадлежности. Так, нейронная сеть будет понимать, что одна область чисел – текст Пушкина, а близко к ней, но уже совсем другая – текст Лермонтова, и так далее.

Недостаток применения Embedding – большие требования к вычислительным мощностям. Требования зависят как от размера текстовой базы, так и от размера Embedding-слоя. Обычно наибольший размер для Embedding-слоя подразумевает расчёт на наибольшую точность сети, хотя это тоже зависит от конкретной задачи.

* 1. **Выводу по разделу**

В ходе выполнения работы был рассмотрен язык графического описания UML.

Были построены следующие диаграммы:

* диаграмма вариантов использования «as is» для пользователя;
* диаграмма вариантов использования «as to be» для пользователя, сервера и программы;
* диаграмма деятельности «as is» с точки зрения работы сотрудников;
* диаграмма деятельности «as is» с точки зрения работы программы;
* диаграммы деятельности «as to be» с точки зрения работы сотрудников;
* диаграммы деятельности «as to be» с точки зрения работы программы;
* диаграмма последовательности «as is» для пользователя, программы и сервера;
* диаграмма последовательности «as to be» для пользователя, чат-бота, программы и сервера.

С помощью данных диаграмм можно показать различия между текущим состоянием работы сотрудников и как облегчится работа после введения чат-бота в эксплуатацию, а также можно наглядно увидеть области применения системы и взаимодействие между объектами внутри нее.

1. **Практическая реализация**

**3.1 Выбор среды разработки**

В качестве среды разработки был выбран Colaboratory, или просто Colab.

Блокнот Colab – это бесплатная интерактивная облачная среда для работы с кодом от Google, позволяет писать и выполнять код прямо в браузере. В его основе блокнот Jupyter для работы с языком Python, для использования базы используется Google Drive, а не хранилище компьютера. Главная особенность «Колаборатории» – бесплатные мощные графические процессоры GPU и тензорные процессоры TPU, благодаря которым можно заниматься не только базовой аналитикой данных, но и более сложными исследованиями в области машинного обучения. С тем, что CPU вычисляет часами, GPU или TPU справляются за минуты или даже секунды. Сам

Colab предоставляет графические процессоры, хранилище данных, не требует дополнительных настроек и позволяет дать доступ к документам другим людям и работать с ними одновременно, и все это полностью бесплатно.

Сервис написан на языке программирования Python.

**3.2 Практическая реализация чат-бота**

Чат-бот – это сервис, автоматизирующий работу сотрудника в управлении клиентского сервиса и позволяющий сократить время ожидания клиента, а также существенно сократить вероятность потери чата.

Рассмотрим фрагмент программного кода подключения всех необходимые инструменты: библиотеки целиком, различные модули, функции, нейронные слои.

А также базу данных для ее обработки и дальнейшего обучения нейронной сети, отрисовки графиков, разбиение на тренировочную и тестовую выборки, фрагмент кода представлен на рисунках 3.1 и 3.2.

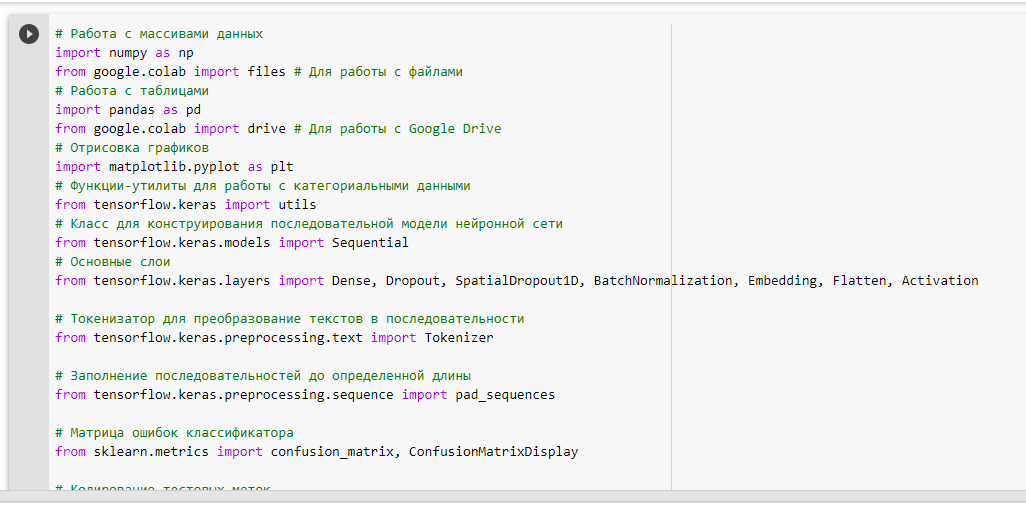


Рисунок 3.1 – Фрагмент кода подключения библиотек

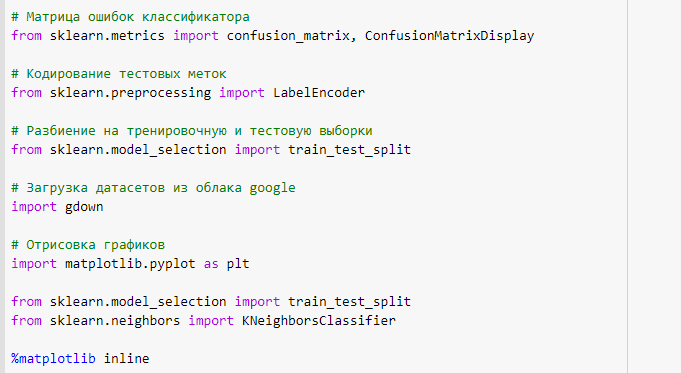


Рисунок 3.2 – Фрагмент кода подключенных библиотек (продолжение)

Представление данных в файле датасета (текст, категория и ответственный по обращению) показано на рисунке 3.3.

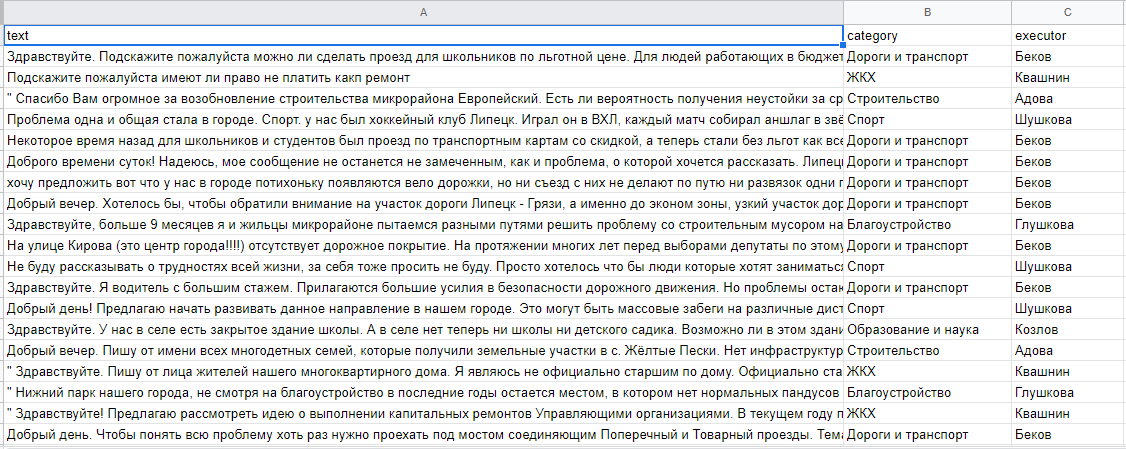


Рисунок 3.3 – Представление данных в датасете

Загрузка датасета и подготовка данных происходит через файл Excel, загруженный на Google Drive. Библиотека Pandas поможет прочесть данные в виде таблицы, чтобы работа с ними была удобнее. Фрагмент кода загрузки датасета и его представление показано на рисунке 3.4.

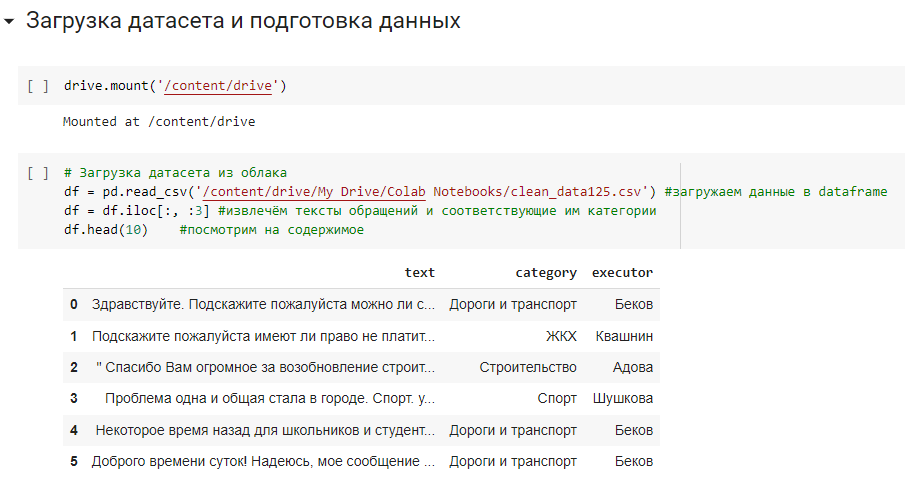


Рисунок 3.4 – Загрузка датасета

Затем необходимо выделить часть текста по частоте обращений слов. Выделяем объем 30000 для токенизатора и долю проверочной выборки в 20%. Фрагмент кода выделения лимита показан на рисунке 3.5.

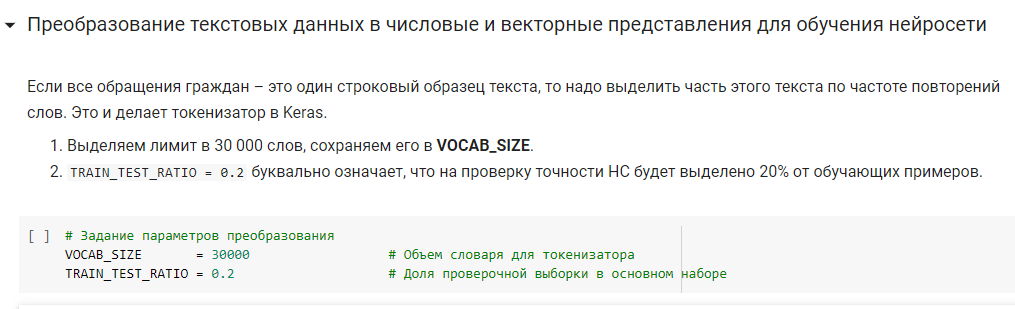


Рисунок 3.5 – Выделение части текста по частоте обращений слов

Tokenizer составит на основе текста обращений граждан словарь частотности с ограничением до индекса 30 000 (значение VOCAB\_SIZE). Тогда каждое слово получит свой индекс в зависимости от частоты его повторений, а до этого из текста будут убраны все символы, все заглавные буквы станут строчными и будут разделены лишь пробелами. Как вы уже знаете, все слова, не попавшие в диапазон VOCAB\_SIZE будут признаны неизвестными словами и получат такое же название и индекс 1.

Сам словарь формируется для токенайзера функцией fit\_on\_texts(). Функция word\_index\_items() поможет образовать тот же словарь в виде пар «слово : индекс частоты повторений слова». Фрагмент кода построения словаря показан на рисунке 3.6.

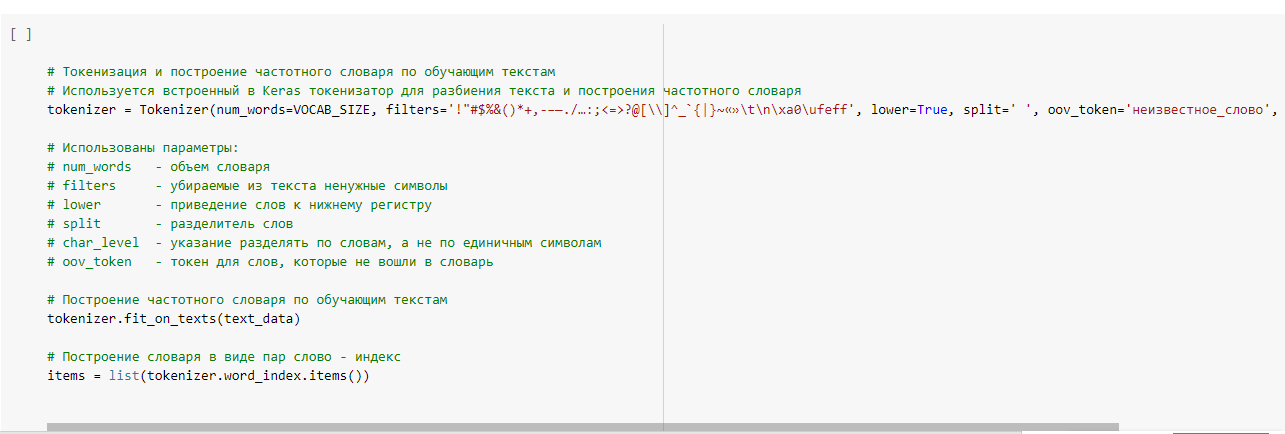


Рисунок 3.6 – Фрагмент кода построения словаря

В библиотеке Sklearn есть очень полезный инструмент – LabelEncoder. Его используют в тех случаях, когда работают с категориальными данными. В нашей таблице есть множество категорий обращений граждан, к примеру «Спорт», «ЖКХ» или «Строительство». Нейронная сеть будет выявлять принадлежность обращения гражданина к одной из таких категорий.

LabelEncoder преобразует весь список категорий текстового типа (class\_data) в список индексов от 0 и до последней категории по счету. Если какая-либо категория встречается повторно, то она все равно сохраняет тот индекс, что первоначально ей присвоен. Так, сколько раз бы не встретилась категория «Спорт», если ее индекс 3, то он так и останется 3.

Всего категорий (или классов) получилось 22. Это значение сохранено в переменной CLASS\_COUNT.

Размер словаря превышает лимит в 30 000, что нормально, поскольку лимит задан для частоты повторений, а не для общего количества слов. Например, союз «и» может иметь индекс 2, но встречаться очень большое количество раз. Еще заметен очень странный вид названия класса «Благоустройство». Иногда сама кодировка текста может слегка сбиться таким вот образом, но на работе нейронной сети это никак не сказывается. Фрагмент кода преобразования категорий обращений показан на рисунке 3.7.

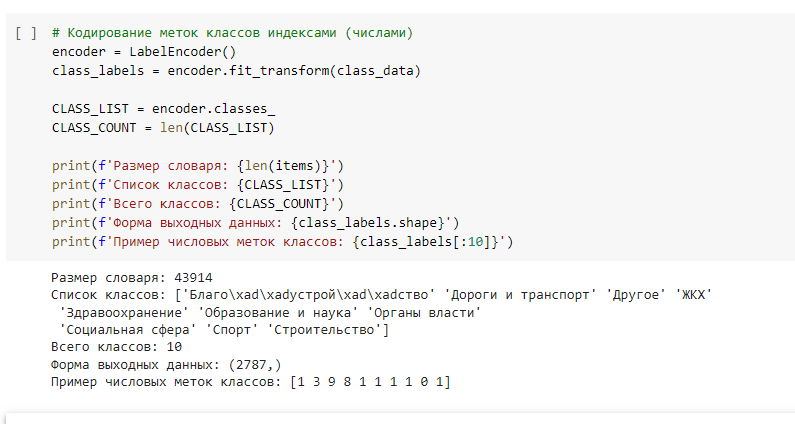


Рисунок 3.7 – Фрагмент кода преобразования категорий обращений

Словарь частотности повторений уже составлен.

Используя функцию токенайзера texts\_to\_sequences() получен текст для обучения в виде одних только индексов (чисел).

Переменная x\_data задана для подготовки данных к самому обучению, потому же данные внутри нее сразу преобразуются в numpy-массив через np.array():

В блоке кода ниже текст обращений также преобразуется в матрицу (texts\_to\_matrix) из значений 0 и 1 для изученного ранее способа Bag Of Words. Эти данные сохраняются в переменной x\_data\_01.

Верные ответы (категории обращений) были переведены LabelEncoder в формат индексов. Но нейронные сети хорошо работают именно с нормализованными данными. Формат от 0 до 1 – лучшая нормализация. Однако и ряд индексов категорий – это не полноценный текст, а просто слова и словосочетания (метки классов).

Для преобразования категорий ответов для подачи в сеть чаще всего используют метод utils.to\_categorical. Он дает вектор из нулей и одной единицы (положение верного ответа) для каждой категории в отдельности. Фрагмент кода преобразования текста в последовательности индексов показан на рисунке 3.8.

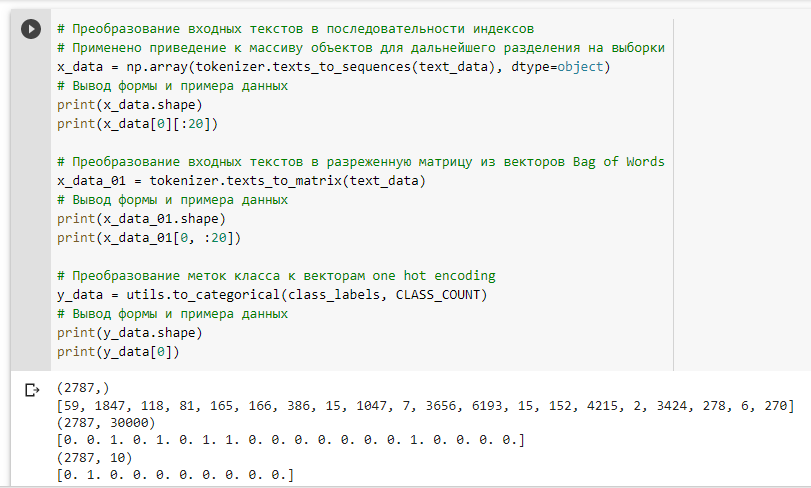


Рисунок 3.8 – Фрагмент кода преобразования текста в последовательности индексов

Создание обучающей и тестовой выборок. На данный момент располагаем x\_data – общим массивом чисел для обучения.

Этот массив необходимо поделить на обучающую выборку и тестовую. Метод train\_test\_split() поможет провести границу деления, результат которого для границы обучающей выборки сохраняют переменные idx\_train и idx\_test – для тестовой.

В параметрах метода указано:

* range(len( )) для x\_data – диапазон длины всего массива для обучения;
* stratify = y\_data – сопоставление тренировочных и тестовых данных с ответами к ним;
* test\_size = train\_test\_ratio – то самое значение деления (train\_test\_ratio = 0.2) общего массива на 80% (тренировочная выборка) к 20% (тестовая выборка).

Поместим idx\_train, idx\_test в диапазон x\_data (x\_data[idx\_train], x\_data[idx\_test]). Так получаем уже финальную выборку. Их можно подать на обучение сети. То же самое сделать для x\_data\_01, чтобы получить выборки для способа Bag Of Words.

В конце важно проверить размерности каждой выборки, чтобы правильно задать гиперпараметры обучения НС. Это можно сделать с помощью shape().

Происходит получение индексов разделения основного набора на обучающую и тестовую выборки, далее разделение в соответствии с полученными индексами, а затем проверка результата.

В итоге получаем цифру 2229, это и есть размерность обучающей выборки, а 558 – размерность тестовой.

Цифра 30000 обозначает объявленный ранее объем словаря для токенизатора.

Фрагмент кода создания обучающей и тестовой выборок показан на рисунке 3.9.

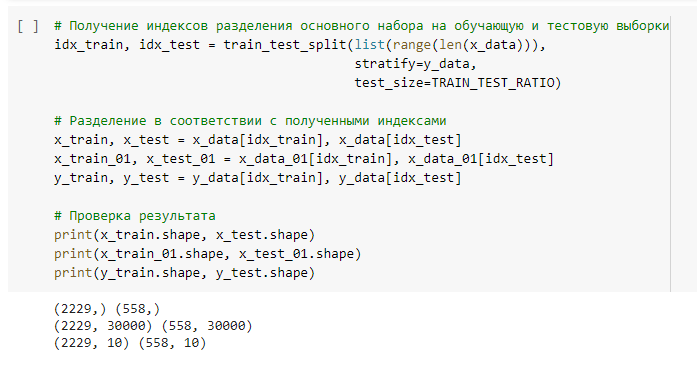


Рисунок 3.9 – Фрагмент кода создания обучающей и тестовой выборок

Функции, указанные на рисунках 3.10 и 3.11 собирают параметры обучения модели воедино (компиляция), оценивают точность уже обученной сети и выводят графики процесса обучения и точности. Две первые функции последовательно вызываются в третьей – compile\_train\_eval\_model(). Пример кода функции обучения и компиляции модели нейронной сети указаны на рисунках 3.10, 3.11, 3.12.



Рисунок 3.10 – Фрагмент кода функции обучения и компиляции модели нейронной сети

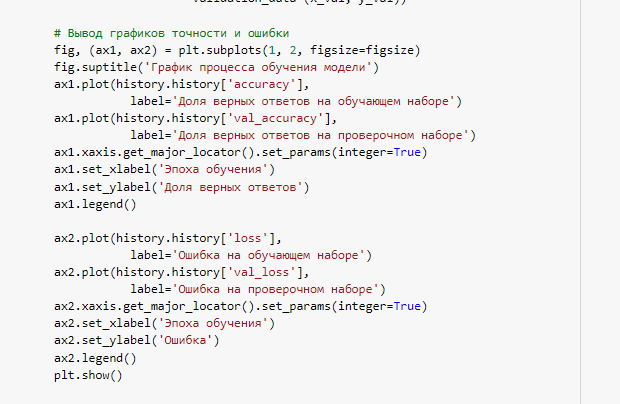


Рисунок 3.11 – Фрагмент кода вывода графиков точности и ошибки

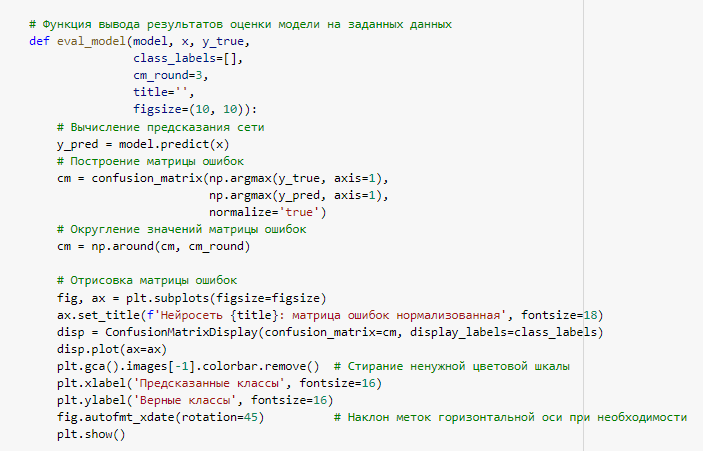


Рисунок 3.12 – Фрагмент кода функции вывода результатов оценки модели

Фрагмент кода функции обучения и оценки модели нейронной сети показан на рисунке 3.13.



Рисунок 3.13 – Фрагмент кода компиляции и обучения модели, вывод результатов

Использованная модель – полносвязная нейронная сеть с применением Bag Of Words. Исходя из этого, имя модели model\_text\_bow\_dense состоит из 7 слоев, из которых 4 слоя – полносвязные (Dense).

Размеры выборок для Bag Of Words x\_train\_01 и x\_test\_01 – это 8047 х 30 000 и 2012 х 30 000 соответственно. Здесь 30 000 – параметр размера созданного ранее словаря частотности. Другими словами – количество значений, которое сеть должна получить на вход. Значение было сохранено в переменной VOCAB\_SIZE.

Каждый Dense-слой сети включает по 100 нейронов и функцию активации relu. Кроме слоя выхода там должно получиться число нейронов в соответствии с количеством определяющихся категорий (10, это число сохранено в CLASS\_COUNT). Функция активации тоже отлична (здесь softmax).

В качестве функции активации для входного слоя и следующих нескольких полносвязных слоев в данной модели используется ReLu.

Ее формула показана на формуле 3.1. График этой функции показан на рисунке 3.14.

|  |  |
| --- | --- |
| , | ((3.1) |

где x – входные данные.

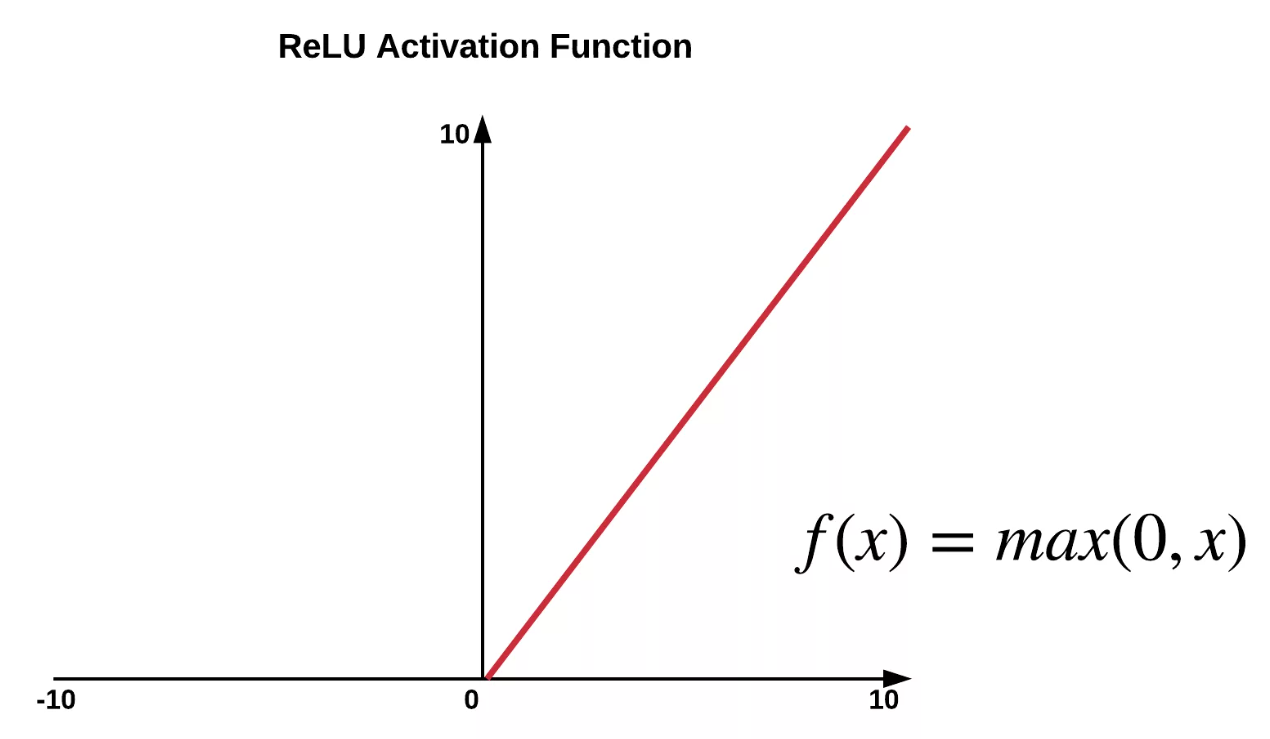


Рисунок 3.14 – График функции активации ReLu

Это наиболее часто используемая функция активации при глубоком обучении. Данная функция возвращает 0, если принимает отрицательный аргумент, в противном случае, функция возвращает само число.

Преимущества данной функции и ее популярность заключается в том, что быстро и просто считается производная, при положительных значениях будет возвращена 1, при отрицательных будет возвращен 0, а также при работе с большой нейронной сетью с множеством нейронов, использование, например, сигмоидной функции в качестве активационный влечет активацию почти всех нейронов, что может сказаться на производительности обучения модели.

Если же использовать ReLU, то количество включаемых нейронов станет меньше, в силу характеристик функции, и сама сеть станет легче.

Но у данной функции есть очень важный недостаток, его название – проблема умирающего ReLu. Так как часть производной функции равна нулю, то и градиент для нее будет нулевым, а то это значит, что веса не будут изменяться во время спуска и нейронная сеть перестанет обучаться [9].

В качестве последнего слоя указана функция softmax, или же нормализованная экспоненциальная функция. Она представляет собой обобщение логистической функции на несколько измерений. Он используется в полиномиальной логистической регрессии и часто используется в качестве последней функции активации нейронной сети для нормализации выходных данных сети к распределению вероятностей по прогнозируемым выходным классам. Функция softmax принимает в качестве входных данных вектор z из K действительных чисел и нормализует его в распределение вероятностей, состоящее из K вероятностей, пропорциональных экспоненциалам входных чисел. То есть до применения softmax некоторые компоненты вектора могли быть отрицательными или больше единицы; и может не равняться 1, но после применения softmax каждый компонент будет в промежутке (0,1), а компоненты в сумме дадут 1, так что их можно интерпретировать как вероятности. Кроме того, более крупные входные компоненты будут соответствовать более высоким вероятностям.

Ее формула показана на формуле 3.2. График этой функции показан на рисунке 3.14.

|  |  |
| --- | --- |
| , | ((3.2) |

где x – вектор-столбец,

k – количество классов,

i – количество нейронов.

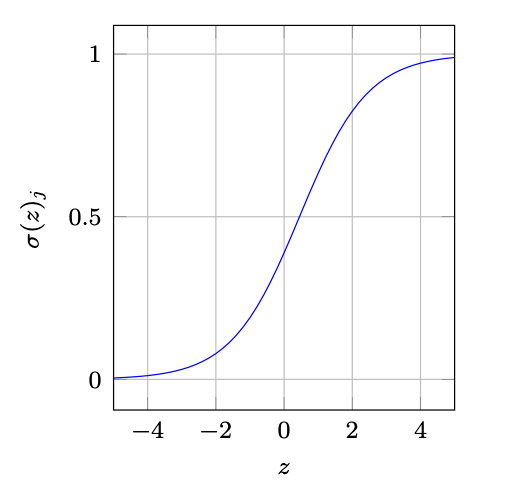


Рисунок 3.14 – График функции активации softmax

Фрагмент кода создания модели и слоев для нейронной сети с функцией активации ReLu показан на рисунке 3.15.



Рисунок 3.15 – Фрагмент кода создания модели и слоев для нейронной сети на модели Bag Of Words c функцией активации ReLu

Параметры модели нейронной сети показаны на рисунке 3.16.

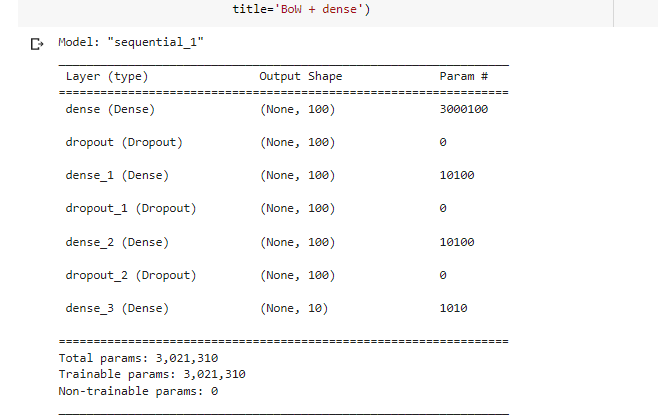


Рисунок 3.16 – Параметры модели нейронной сети модели Bag Of Words c функцией активации ReLu

Процесс обучения нейросети для модели Bag Of Words показан на рисунке 3.17.

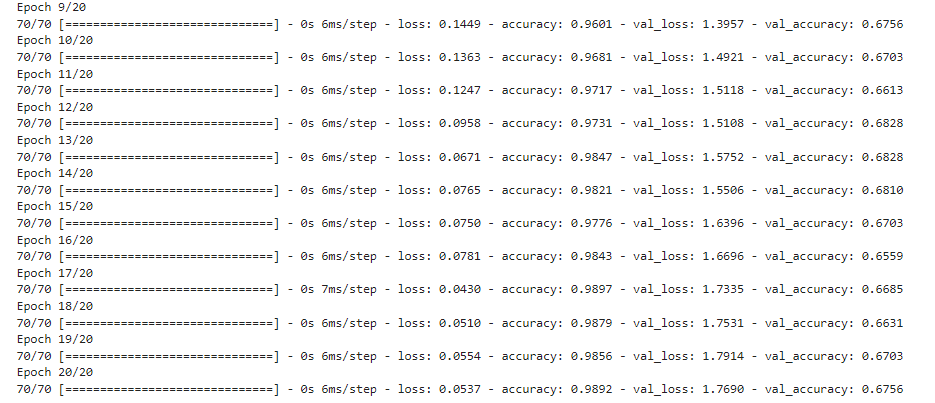


Рисунок 3.17 – Процесс обучения нейронной сети на модели Bag Of Words c функцией активации ReLu

Также можно посмотреть на процесс обучения нейронной сети в виде графика доли верных ответов к эпохе обучения. Синим цветом показана доля верных ответов на обучающем наборе, оранжевым доля верных ответов на проверочном наборе, информация показана на рисунке 3.18.

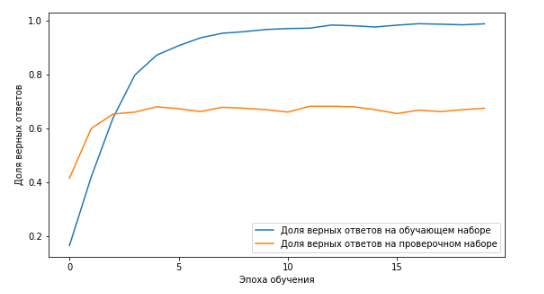


Рисунок 3.18 – График доли верных ответов в процессе обучения нейросети на модели Bag Of Words c функцией активации ReLu

Предсказанные классы, процент отнесения нейросети к каждому классу, и средняя точность распознавания на модели Bag Of Words показаны на рисунке 3.19.

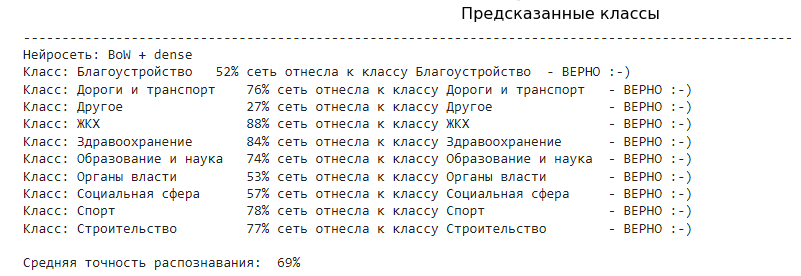


Рисунок 3.19 – Предсказанные классы в процессе обучения нейросети на модели Bag Of Words c функцией активации ReLu

На рисунке 3.20 показан график процесса обучения нейросети в соотношении эпохи обучения и доли ошибки на проверочном и обучающем наборах с используемой функцией ReLu.

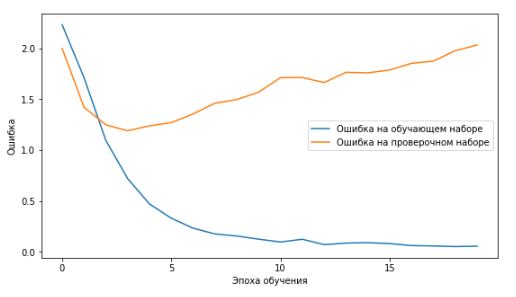


Рисунок 3.20 – График ошибки сети на обучающем и проверочном наборах модели Bag Of Words c функцией активации ReLu

После запуска нейросети на архитектуре BagOfWords + Dense с функцией активации ReLu и на обучающей, и на проверочной выборках процесс обучения проходил с положительной тенденцией, но ошибка сети немного росла на проверочной, что может говорить о необходимости немного поменять архитектуру НС. Статистика распределения по классам показала, что НС довольно хорошо научилась предсказывать 7 из 10 категорий, мы получили среднюю точность работы сети – 69%, что показывает хороший результат.

Чтобы изменить архитектуру нейронной сети, необходимо изменить функцию активации.

В качестве функции активации для входного слоя возьмем sigmoid (сигмоида).

Ее формула показана на формуле 3.3. График этой функции показан на рисунке 3.21.

|  |  |
| --- | --- |
| , | ((3.3) |

где s – входные данные.

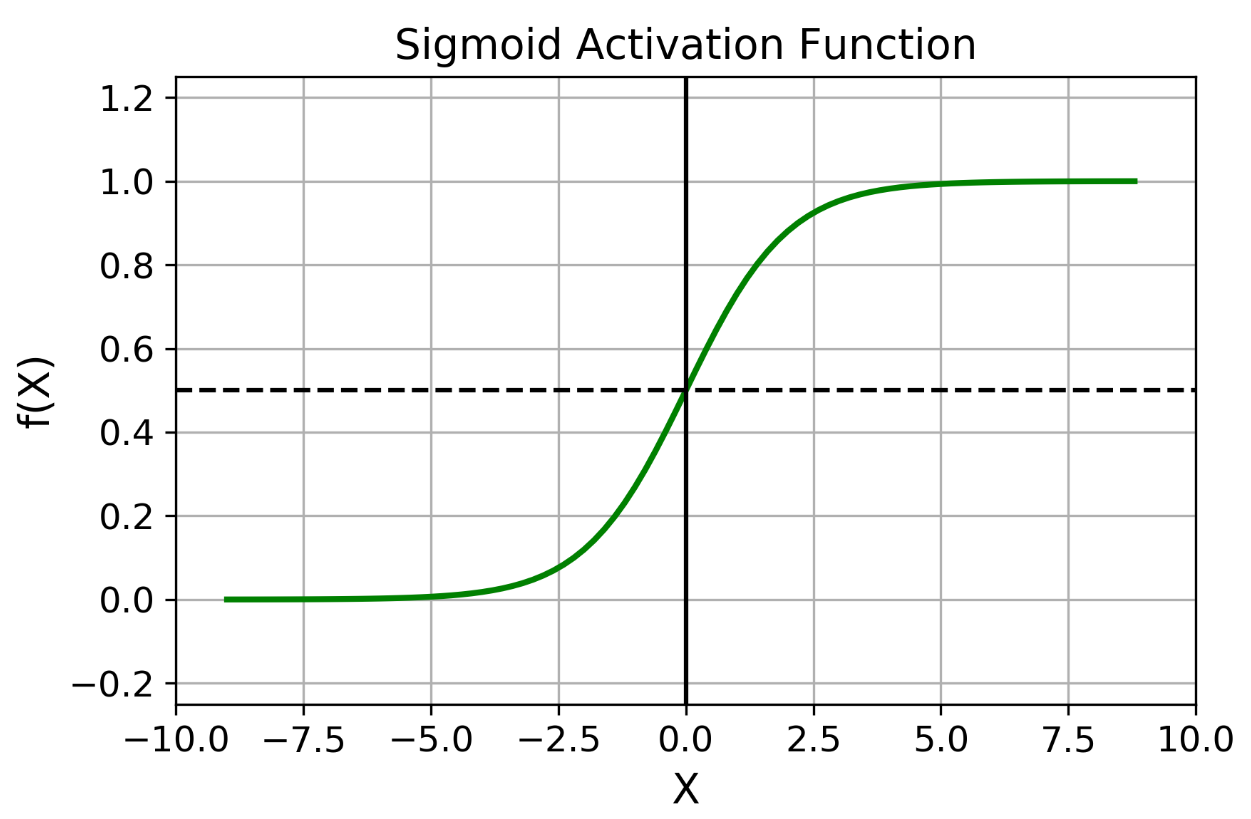


Рисунок 3.21 – График функции активации sigmoid

Эта функция относится к классу непрерывных функций и принимает на входе произвольное вещественное число, а на выходе дает вещественное число в интервале от 0 до 1. В частности, большие (по модулю) отрицательные числа превращаются в ноль, а большие положительные – в единицу.

Исторически сигмоида находила широкое применение, поскольку ее выход хорошо интерпретируется, как уровень активации нейрона: от отсутствия активации (0) до полностью насыщенной активации (1).

В процессе обратного распространения ошибки данный градиент умножается на общий градиент. Следовательно, если локальный градиент будет крайне мал, то он будет обнулять общий. В результате чего, сигнал почти не будет проходить через нейрон к его весам и рекурсивно к его данным.

Если исходные веса имеют слишком большие значения, большинство нейронов перейдет в состояние насыщения, в результате чего сеть будет плохо обучаться. Также сигмоида хорошо показывает себя в задачах классификации.

Фрагмент кода создания модели и слоев для нейронной сети с функцией активации sigmoid показан на рисунке 3.22.



Рисунок 3.22 – Фрагмент кода создания модели и слоев для нейронной сети на модели Bag Of Words с функцией активации sigmoid

Параметры модели нейронной сети показаны на рисунке 3.23.

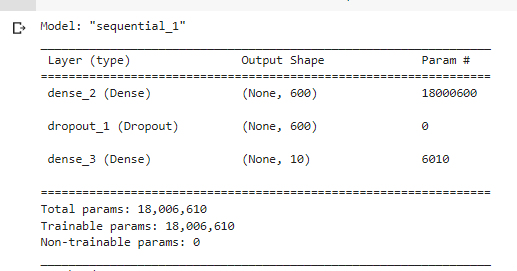


Рис 3.23 – Параметры модели нейронной сети модели Bag Of Words с функцией активации sigmoid

Процесс обучения нейросети для модели Bag Of Words c функцией активации sigmoid показан на рисунке 3.24.

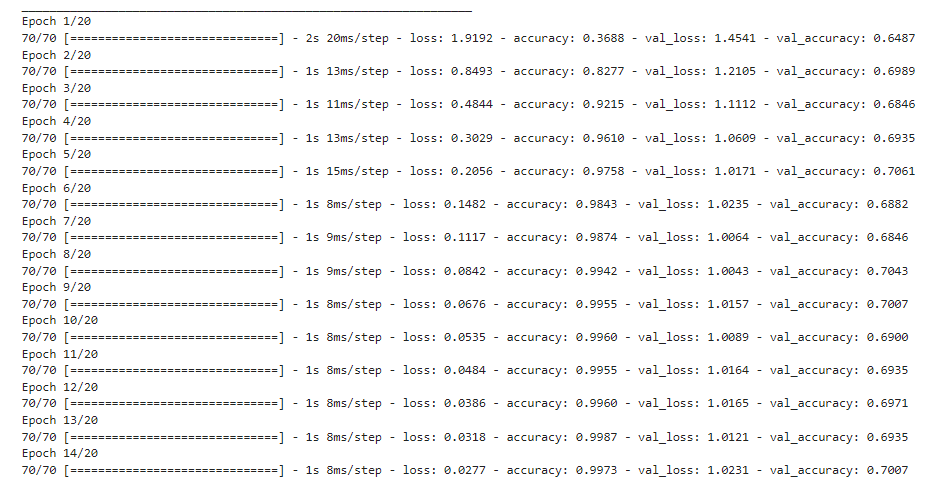


Рисунок 3.24 – Процесс обучения нейронной сети на модели Bag Of Words с функцией активации sigmoid

Также можно посмотреть на процесс обучения нейронной сети в виде графика доли верных ответов к эпохе обучения, информация показана на рисунке 3.25.

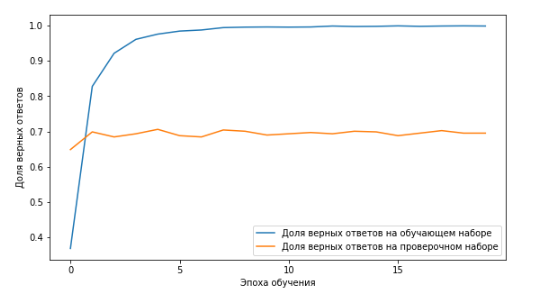


Рисунок 3.25 – График доли верных ответов в процессе обучения нейросети на модели Bag Of Words с функцией активации sigmoid

Предсказанные классы, процент отнесения нейросети к каждому классу, и средняя точность распознавания на модели Bag Of Words с функцией активации sigmoid показаны на рисунке 3.26.

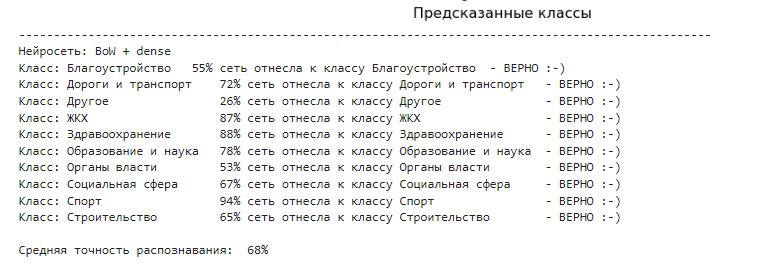


Рисунок 3.26 – Предсказанные классы в процессе обучения нейросети на модели Bag Of Words с функцией активации sigmoid

На рисунке 3.27 показан график процесса обучения нейросети в соотношении эпохи обучения и доли ошибки на проверочном и обучающем наборах с используемой функцией sigmoid.

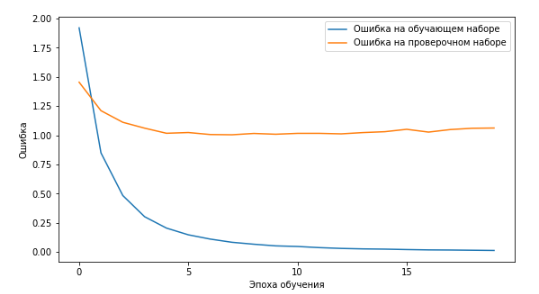


Рисунок 3.27 – График ошибки сети на обучающем и проверочном наборах модели Bag Of Words с функцией активации sigmoid

После запуска нейросети на архитектуре BagOfWords + Dense с функцией активации Sigmoid и на обучающей, и на проверочной выборках процесс обучения проходил с положительной тенденцией, но, в отличие от функции ReLu ошибка сети не росла, а наоборот, снижалась и на проверочной, и на обучающей выборках.

Получена средняя точность работы сети – 68%, что показывает аналогичный результат. Но, из-за того, что ошибка сети снижалась и осталась на моменте, близком к 1, а не осталась выше 2, как в случае с ReLu, то при дальнейшем использовании обучение будет проходить с помощью функции сигмоиды.

Embedding + Dense. Библиотека matplotlib помогает в визуальной оценке данных. На рисунке 3.28 можно увидеть, что наибольшее количество обучающих примеров сконцентрировано в промежутке от 0 (данные отсутствуют) до 200. Окончательно же концентрация примеров спадает от 400 к 600.

Это значит, что примеров обращений объемом выше 600 в тренировочных данных очень мало, а больше всего либо пустых примеров («нет данных»), либо тех, объем которых в районе 100 значений в примере.

Вложенный цикл for – способ создать необходимые значения координат для гистограммы. Они сохранены в seq\_list (по ним получается кол-во столбцов графика и их высота). Далее строится сам график с помощью plt.hist(), где в параметрах число 50 – число столбцов на всем графике.

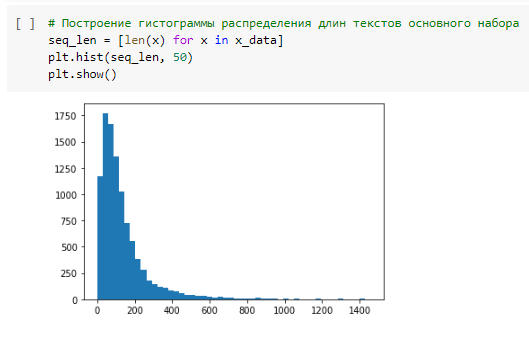


Рисунок 3.28 – Построение гистограммы распределения длин текстов основного набора

На рисунке 3.29 предложен следующий подход: ограничить размер каждого примера. В данном случае до 400 слов. Остальное в каждом примере будет просто отброшено.

Сохраняем это значение в seq\_max\_len. Создаем переменные x\_train\_clip и x\_test\_clip для выборок и укрощаем длину примеров в них с помощью pad\_sequences().

Функция pad\_sequences() не только обрезает примеры, но и дополняет их до нужного размера, если они короче. В результате вместо размерностей 8047 х 30 000 и 2012 х 30 000 получаем 8047 х 400 и 2012 х 400 для каждой выборки.

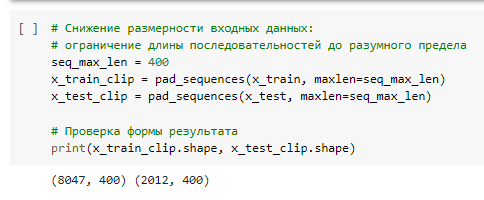


Рисунок 3.29 – Пример кода для ограничения размера каждого примера до 400 слов

На рисунке 3.30 показан пример кода архитектуры модели для Embedding. Архитектура включает один входной слой Embedding, два Dense-слоя для полносвязной обработки значений и несколько слоев регуляризации и нормализации.

Также используется дополнительный гиперпараметр слоя Embedding input\_length, равный seq\_max\_len, то есть значению 400.

Этот параметр позволяет слою принять для обработки данные из выборок x\_train\_clip и x\_test\_clip.



Рисунок 3.30 – Пример кода архитектуры модели для Embedding

Параметры модели нейронной сети для модели Embedding показаны на рисунке 3.31.

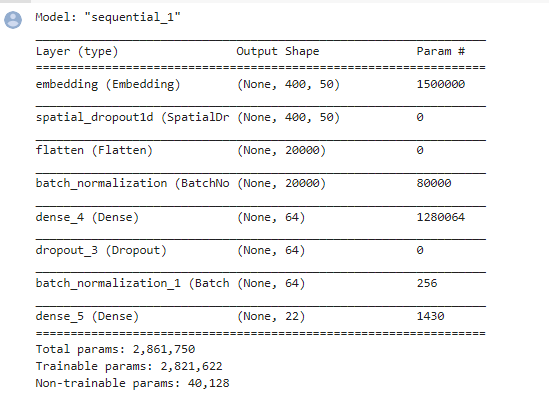


Рисунок 3.31 – Параметры модели нейронной сети для модели Embedding

Процесс обучения нейросети для модели Embedding показан на рисунке 3.32.

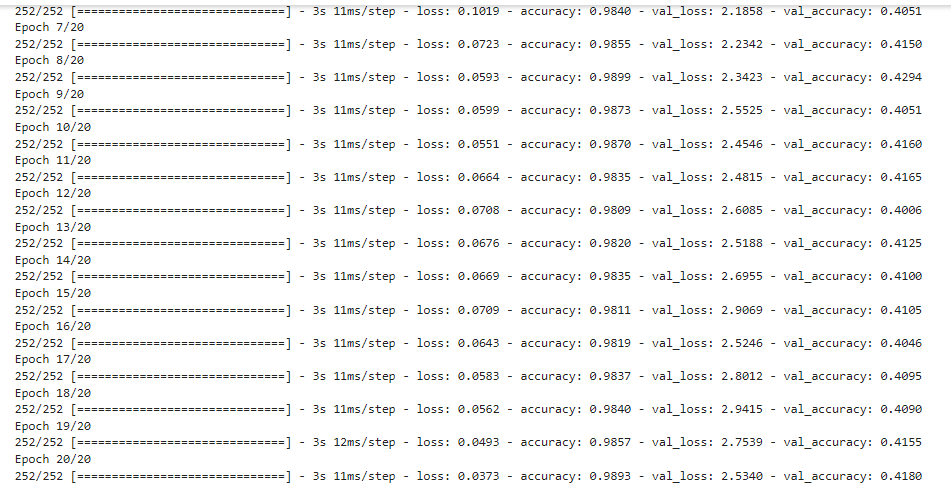


Рисунок 3.32 – Процесс обучения нейросети для модели Embedding

Также можно посмотреть на процесс обучения нейронной сети в виде графика доли верных ответов к эпохе обучения, информация показана на рисунке 3.33.

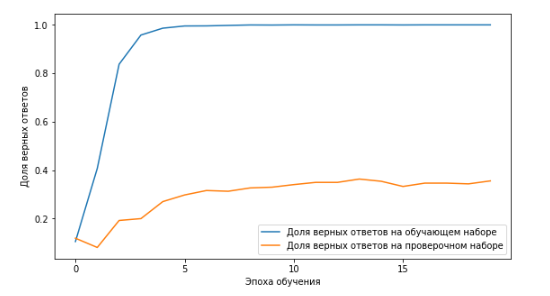


Рисунок 3.33 – График доли верных ответов в процессе обучения нейросети на модели Embedding

Предсказанные классы, процент отнесения нейросети к каждому классу, и средняя точность распознавания на модели Embedding показаны на рисунке 3.34.

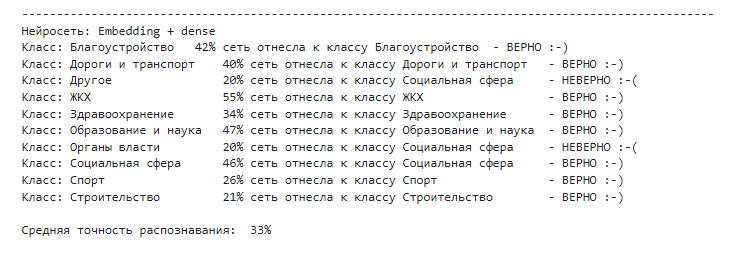


Рисунок 3.34 – Предсказанные классы в процессе обучения нейросети на модели Embedding

Статистика распределения по классам показала, что нейросеть очень слабо научилась предсказывать категории «Дороги и транспорт», «Здравоохранение» и «ЖКХ». Средняя точность работы сети составила 33% – оценка неудовлетворительно. В обработке базы обращений граждан наилучший результат показала полносвязная сеть Bag Of Words.

Но не стоит недооценивать Embedding: использование данного слоя в нейросети просто не подошло под конкретный тип задачи. Есть ряд задач, где Embedding, напротив, справится лучше, чем Bag Of Words, например – классификация тексов писателей. Следовательно, в работе будет использоваться модель нейросети с применением Bag Of Words.

Так как понятие классификации обращений подразумевает еще назначение его на исполнителя, то необходимо обучить еще одну нейросеть, которая будет соотносить текст обращения и сотрудника, работающего с этим.

Нейросеть для классификации обращений по исполнителям будет применять модель Bag Of Words и иметь аналогичные настройки и архитектуру.

Фрагмент кода создания модели и слоев НС показан на рисунке 3.35.



Рисунок 3.35 – Пример кода архитектуры модели нейросети для классификации исполнителей с применением Bag Of Words

Параметры модели нейронной сети для модели Bag Of Words показаны на рисунке 3.36.

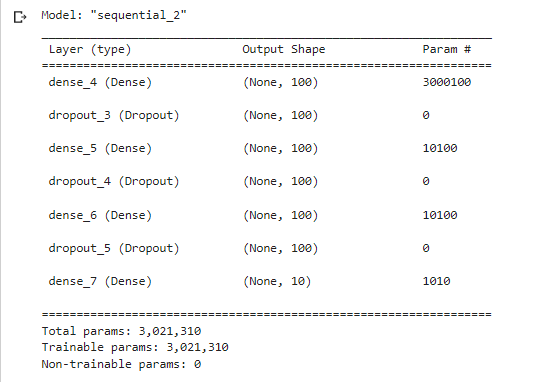


Рисунок 3.36 – Параметры модели нейронной сети для модели Bag Of Words

Также можно посмотреть на процесс обучения нейронной сети в виде графика доли верных ответов к эпохе обучения, информация показана на рисунке 3.37.

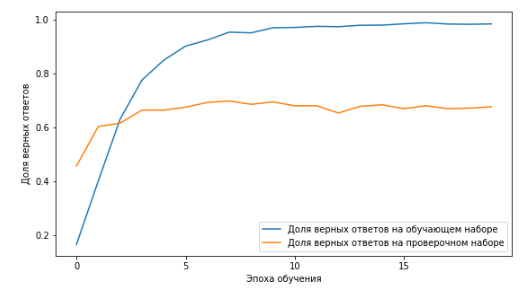


Рисунок 3.37 – График доли верных ответов в процессе обучения нейросети на модели Bag Of Words

Предсказанные классы, процент отнесения нейросети к каждому классу, и средняя точность распознавания на модели Bag Of Words показаны на рисунке 3.38.

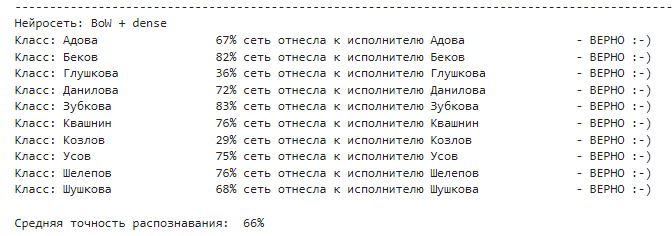


Рисунок 3.38 – Предсказанные классы в процессе обучения нейросети на модели Bag Of Words

Для полноценной дальнейшей работы с нейросетью необходимо вводить в нее текст обращения и получать ответ о его категории и исполнителе. Для этого нужно подключить его к мессенджеру Telegram как чат-бота.

Для создания потребуется зайти в канал @BotFather, далее написать команду /newbot и написать имя нашего бота. В итоге получаем сообщение о регистрации бота с этим названием и токен API для подключения к нему. Процесс создания показан на рисунке 3.39. API-токен на рисунке замазан в целях безопасности.

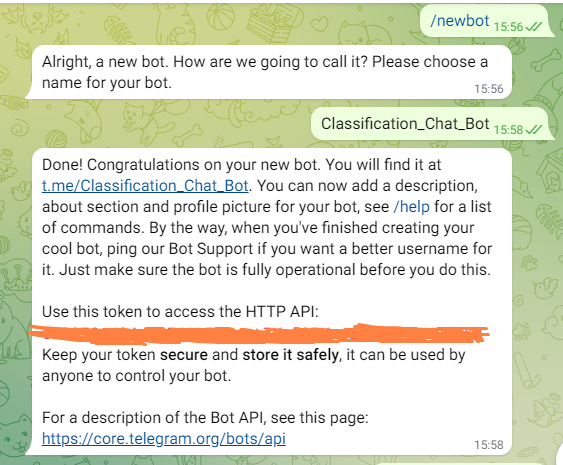


Рисунок 3.39 – Процесс создания бота в Telegram

В качестве обёртки взята библиотека python-telegram-bot. Она предоставляет чистый интерфейс Python для Telegram Bot API. Он совместим с версиями Python 3.6. и выше.

Пример кода для подключения библиотеки и результата показан на рисунке 3.40.

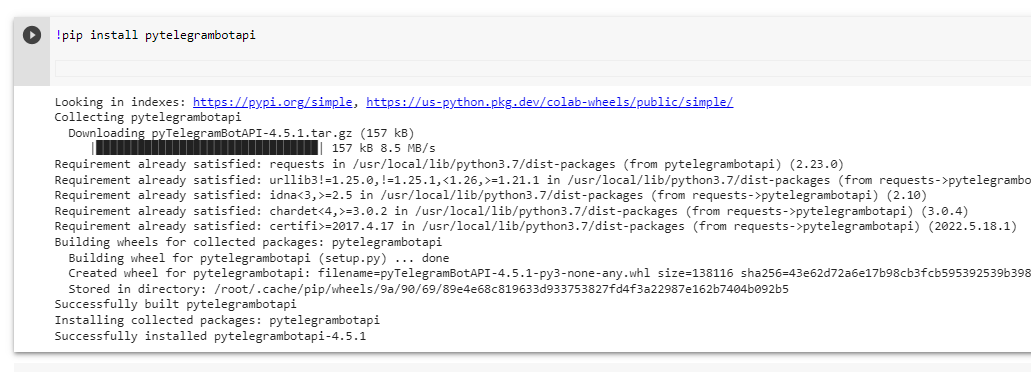


Рисунок 3.40 – Пример кода подключения библиотеки

После установки нужно написать «основу», которая связывает программу и чат-бота телеграмма. Импортируем необходимые модули и прописываем токен API, на рисунке он замазан в целях безопасности. Пример кода показан на рисунке 3.41.

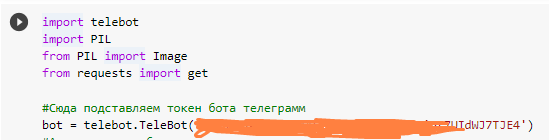


Рисунок 3.41 – Импорт модулей и прописывание токена

Далее напишем 2 обработчика команд. Это callback-функции, которые будут вызываться тогда, когда будет получено обновление. Напишем две таких функции для команды /start и для обычного любого текстового сообщения.

Боты не могут инициировать чаты с пользователями. Пользователь должен либо добавить их в группу, либо сначала отправить им сообщение.

В качестве аргументов туда передаются один параметр: message. Этот параметр содержит данные о пришедшем сообщении. Пример кода обработчиков команд показан на рисунке 3.42.

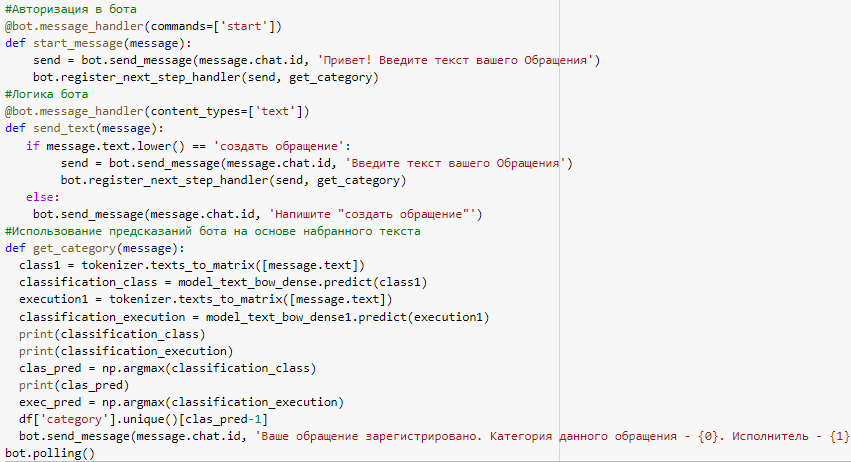


Рисунок 3.42 – Пример кода обработчиков команд

Далее необходимо проверить работу программы и нейросети. Для этого запустим код и зайдем в чат-бот телеграмм. Для начала работы нужно начать кнопку Старт внизу экрана. Затем бот предложит ввести текст обращения. Для примера вводим вопрос: «Когда починят дороги в Новосибирске на площади Ленина?». Получаем ответ, что наше Обращение зарегистрировано, категория нашего Обращения – Дороги и транспорт. На рисунке 3.43 показан текст обращения и результат работы программы.

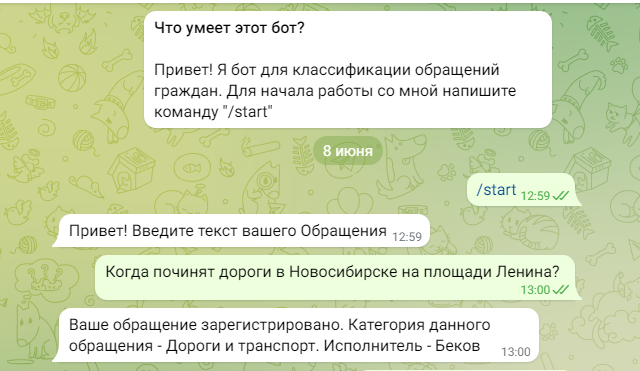


Рисунок 3.43 – Результат работы программы

Как видно из рисунка, бот верно определил категорию обращения – Дороги и транспорт, а также исполнителя – Беков. В качестве примера напишем текст еще одного обращения. Результат работы программы на другом примере показан на рисунке 3.44.

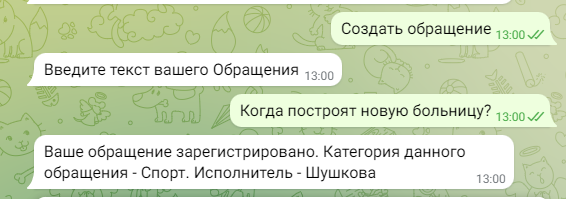


Рисунок 3.44 – Результат работы программы на другом примере

Как видно из рисунка 3.44, нейросеть неверно определила категорию обращения и исполнителя. В вопросе сказано, когда построят новую больницу, следовательно, нейросеть должна определить категорию Здравоохранение или Строительство, но не Спорт. Это связано с тем, что средняя точность распознавания нейросети – 70%, то есть она может ошибаться.

Повысить среднюю точность распознавания можно несколькими способами:

* переделать датасет (что-то добавить, что-то удалить, что-то подредактировать);
* переделать архитектуру нейронной сети;
* дообучить нейросеть в процессе работы.

## **3.3 Оценка экономической эффективности**

Экономическая эффективность – это результат, который может быть получен путем сравнения показателей рентабельности производства по отношению к общим затратам и используемым ресурсам.

Общим показателем является экономическая эффективность системы, характеризующая выполнимость затрат, понесенных при создании и эксплуатации системы. Исходя из этого, можно сформулировать цели, которые преследуют компании, желающие внедрить или модернизировать ИТ-систему. Количественные характеристики этих свойств определяются следующим показателем – каковы будут преимущества внедрения и использования информационной системы.

Рисунки 3.45 и 3.46 соответствуют примерной стоимости заработной платы дежурного на сайтах по поиску работы.

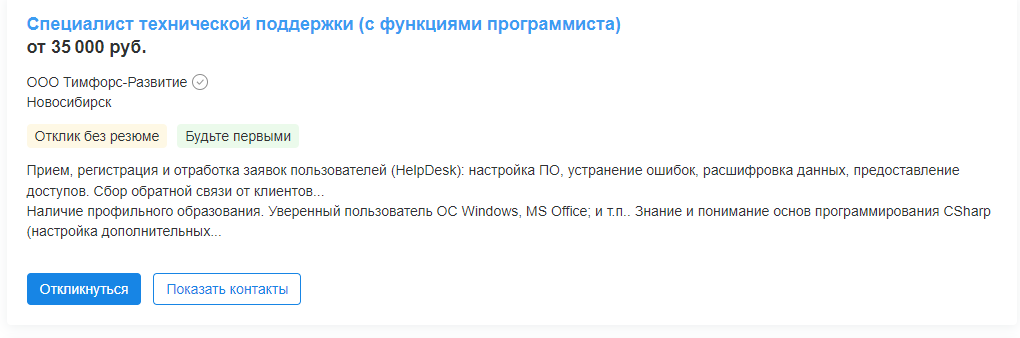


Рисунок 3.45 – Стоимость заработной платы дежурного

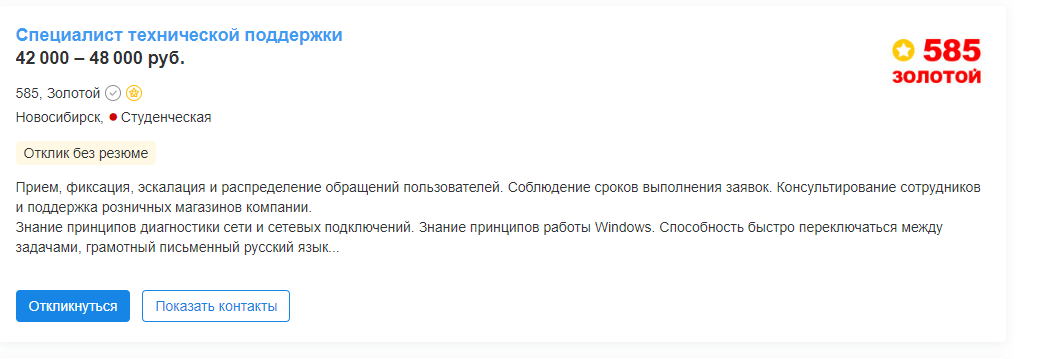


Рисунок 3.46 – Стоимость заработной платы дежурного другой компании

В таблице 1 показаны вводные для рисунка с затратами, на рисунке 3.47 до и после автоматизации бизнес-процесса, данные взяты из рисунков 3.45 и 3.46 как среднее значение.

Таблица 1 – Вводные для рисунка с затратами

|  |  |  |
| --- | --- | --- |
| Для таблицы | | |
| До автоматизации | 0 | 0 |
| После автоматизации | 40000 | 0 |

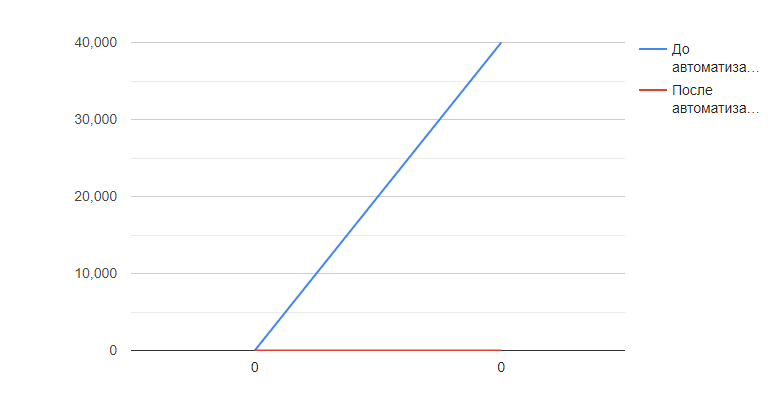


Рисунок 3.47 – Затраты до и после автоматизации на процесс обработки обращений

На этапе эксплуатации до автоматизации были переменные затраты на обработку обращений около сорока тысяч рублей в месяц (средняя заработная плата дежурного), а после автоматизации затраты составляют ноль рублей на тот же самый процесс, так как чат-бот – это автоматизированная система, которой не выплачивается зарплата. Дежурного можно перевести в другой, более важный отдел или группу, не добавляя новое штатное место и не лишая его работы.

Примерная стоимость работы за такой проект оценивается в пятьдесят тысяч рублей. При снижении затрат на сорок тысяч рублей, отображенных на рисунке 3.47, можно сделать вывод, что для окупаемости такого проекта потребуется чуть больше одного месяца, что доказывает, что такой проект очень эффективен и требует введения.

## **3.4 Результаты и выводы**

В данном разделе был рассмотрен бизнес-процесс после внедрения чат-бота, показаны преимущества от внедрения. Далее отображена практическая реализация проекта, были исследованы два варианта подготовки данных, рассмотрены и проанализированы функции активации для обучения нейронной сети, показаны фрагменты кода, представлен алгоритм работы программы. В завершении этого была проведена оценка экономической эффективности проекта.

В результате при исследовании вариантов подготовки данных было выявлено, что в обработке базы обращений граждан наилучший результат показала полносвязная сеть Bag Of Words, соответственно, она была взята в работу и использовалась в разработке нейросети.

При исследовании функций активации для обучения нейронной сети было выявлено, что функция ReLu и Sigmoid показывают практически одинаковую среднюю точность распознавания, но все же при использовании функции сигмоиды ошибка сети не росла, а наоборот падала, и показатель у нее стремится к 1, а не к 2, как у ReLu, именно поэтому в дальнейшей работе при обучении нейросети использовалась функция сигмоиды.

# **ЗАКЛЮЧЕНИЕ**

В первом разделе диссертации был представлен обзор предметной области, описаны основные задачи использования ИИ, группа технологий ИИ, описан искусственный интеллект в работе клиентской и технической поддержки, показаны результаты работы аналогичных программ.

Во втором разделе были подробно описаны цели и задачи, показана структура проекта, при помощи языка UML были построены диаграммы этих бизнес-процессов, описаны методы подготовки текстовых данных для подачи в нейросеть. В конце были подведены выводы по разделу.

В третьем разделе была представлена практическая реализация проекта, исследованы два варианта подготовки данных, рассмотрены и проанализированы функции активации для обучения нейронной сети, показаны фрагменты кода, интерфейс программы и результаты его работы с описанием. В завершении была рассчитана экономическая эффективность проекта.

В ходе выполнения диссертации были исследованы два варианта подготовки данных для обучения нейросети, рассмотрены и проанализированы функции активации, в результате которого было выявлено, что наилучший результат показывает полносвязная сеть Bag Of Words, а функции ReLu и sigmoid показывают одинаковую среднюю точность распознавания, но у второй ошибка сети значительно ниже. В результате, используя эту информацию, был разработан классификатор обращений граждан, автоматизирующий работу специалиста отдела клиентского сервиса или технической поддержки, используемый в мессенджере «Telegram» на языке программирования python.

Подводя итоги выпускной квалификационной работы, можно сделать вывод, что поставленная цель исследования была достигнута, все задачи были полностью выполнены.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1 Всё, что вам нужно знать об ИИ — за несколько минут / [Электронный ресурс] URL: https://habr.com/ru/post/416889/ (Дата обращения 10.05.2022).

2 Перспективы и тенденции развития искусственного интеллекта — за несколько минут / [Электронный ресурс] URL: https://computer-museum.ru/frgnhist/aireview.htm (Дата обращения 10.05.2022).

3 AI for Dinosaurs / [Электронный ресурс] URL: https://medium.com/mechanized/ai-for-dinosaurs-67848a90fce3 (Дата обращения 10.05.2022).

4 Перспективы и тенденции развития искусственного интеллекта [Электронный ресурс] URL: https://computer-museum.ru/frgnhist/aireview.htm (Дата обращения 10.05.2022).

5 Программирование искусственного интеллекта / [Электронный ресурс] URL: https://steptosleep.ru/программирование-искусственного-инт/#12 (Дата обращения 10.05.2022).

6 Искусственный интеллект: технологии и применение / [Электронный ресурс] URL: https://rdc.grfc.ru/2020/12/aitech/#post-934-\_Toc60141806 (Дата обращения 10.05.2022).

7 Artificial intelligence call center: AI’s impact on the customer service / [Электронный ресурс] URL: https://dataconomy.com/2022/03/artificial-intelligence-call-center/ (Дата обращения 10.05.2022).

8 Как искусственный интеллект меняет работу службы поддержки и чатботов / [Электронный ресурс] URL: https://marketer.ua/how-artificial-intelligence-is-changing-how-customer-service-works/ (Дата обращения 10.05.2022).

9 Роль искусственного интеллекта в организации колл центра / [Электронный ресурс] URL: https://itcrumbs.ru/ii-v-koll-tsentre\_33299 (Дата обращения 10.05.2022).

10 Применение Искусственного Интеллекта в работе операторов и при взаимодействии с клиентами / [Электронный ресурс] URL: https://callcenterguru.ru/articles/primeneniye-iskusstvennogo-intellekta-v-rabote-operatorov-i-pri-vzaimodeystvii-s-kliyentami (Дата обращения 10.05.2022).

11 Саппорт IT-продукта, часть 1: чаты, роли сотрудников, боты и отношения с разработкой / [Электронный ресурс] URL: https://skillbox.ru/media/code/support-it-produkta-chast-1/ (Дата обращения 10.05.2022).

12 Как чат-боты меняют службу поддержки / [Электронный ресурс] URL: https://zen.yandex.ru/media/id/622512daa56589209123f0f1/kak-chatboty-meniaiut-slujbu-podderjki-62a09ba0d2d1335b3dbf8e83 (Дата обращения 10.05.2022).

13 Как роботизировать техническую поддержку с помощью машинного обучения / [Электронный ресурс] URL: https://habr.com/ru/company/croc/blog/522072/ (Дата обращения 10.05.2022).

14 Практики реализации нейронных сетей / [Электронный ресурс] URL:https://neerc.ifmo.ru/wiki/index.php?title=Практики\_реализации\_нейронных\_сетей (Дата обращения 10.05.2022).

15 На смену чат-ботам: как мы создали платформу, которая изменит саппорт / [Электронный ресурс] URL: https://vc.ru/tribuna/90989-na-smenu-chat-botam-kak-my-sozdali-platformu-kotoraya-izmenit-sapport (Дата обращения 10.05.2022).

16 Методологии моделирования бизнес-процессов [Электронный ресурс] URL: https://hsbi.hse.ru/articles/metodologii-modelirovaniya-biznes-protsessov-osnovnye-metody-i-metodiki/ (Дата обращения 13.05.2022).

17 Uml [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/UML (Дата обращения 13.05.2022).

# **ПРИЛОЖЕНИЕ А**

**Код программы**

# Работа с массивами данных

import numpy as np

from google.colab import files # Для работы с файлами

# Работа с таблицами

import pandas as pd

from google.colab import drive # Для работы с Google Drive

# Отрисовка графиков

import matplotlib.pyplot as plt

# Функции-утилиты для работы с категориальными данными

from tensorflow.keras import utils

# Класс для конструирования последовательной модели нейронной сети

from tensorflow.keras.models import Sequential

# Основные слои

from tensorflow.keras.layers import Dense, Dropout, SpatialDropout1D, BatchNormalization, Embedding, Flatten, Activation

# Токенизатор для преобразование текстов в последовательности

from tensorflow.keras.preprocessing.text import Tokenizer

# Заполнение последовательностей до определенной длины

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Матрица ошибок классификатора

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

# Кодирование тестовых меток

from sklearn.preprocessing import LabelEncoder

# Разбиение на тренировочную и тестовую выборки

from sklearn.model\_selection import train\_test\_split

# Загрузка датасетов из облака google

import gdown

# Отрисовка графиков

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

%matplotlib inline

drive.mount('/content/drive') #подключение гугл диска

# Загрузка датасета из облака

df = pd.read\_csv('/content/drive/My Drive/Colab Notebooks/clean\_data125.csv') #загружаем данные в dataframe

df = df.iloc[:, :3] #извлечём тексты обращений и соответствующие им категории

df.head(10)    #посмотрим на содержимое

df = df.iloc[:, :3] #извлечём тексты обращений и соответствующие им категории

df.head(10)    #посмотрим на содержимое

# Вывод размерности таблицы

print(df.shape)

# Пример данных из таблицы

print(df.values[0])

# Функция вывода сводки по распределению данных

def data\_summary(df, class\_labels):

    for cls in class\_labels:

        print(f'Количество записей класса {cls}: {df[df.category == cls].shape[0]}')

# Сводка по распределению данных

data\_summary(df, df.category.unique())

# Задание параметров преобразования

VOCAB\_SIZE       = 30000 # Объем словаря для токенизатора

TRAIN\_TEST\_RATIO = 0.2 # Доля проверочной выборки в основном наборе

# Извлечение всех записей-текстов обращений и их меток классов

text\_data = df.text.tolist()

class\_data = df.category.tolist()

executor\_data = df.executor.tolist()

# Токенизация и построение частотного словаря по обучающим текстам

# Используется встроенный в Keras токенизатор для разбиения текста и построения частотного словаря

tokenizer = Tokenizer(num\_words=VOCAB\_SIZE, filters='!"#$%&()\*+,-–—./…:;<=>?@[\\]^\_`{|}~«»\t\n\xa0\ufeff', lower=True, split=' ', oov\_token='неизвестное\_слово', char\_level=False)

# Использованы параметры:

# num\_words   - объем словаря

# filters     - убираемые из текста ненужные символы

# lower       - приведение слов к нижнему регистру

# split       - разделитель слов

# char\_level  - указание разделять по словам, а не по единичным символам

# oov\_token   - токен для слов, которые не вошли в словарь

# Построение частотного словаря по обучающим текстам

tokenizer.fit\_on\_texts(text\_data)

# Построение словаря в виде пар слово - индекс

items = list(tokenizer.word\_index.items())

# Кодирование меток классов индексами (числами)

encoder = LabelEncoder()

class\_labels = encoder.fit\_transform(class\_data)

CLASS\_LIST = encoder.classes\_

CLASS\_COUNT = len(CLASS\_LIST)

print(f'Размер словаря: {len(items)}')

print(f'Список классов: {CLASS\_LIST}')

print(f'Всего классов: {CLASS\_COUNT}')

print(f'Форма выходных данных: {class\_labels.shape}')

print(f'Пример числовых меток классов: {class\_labels[:10]}')

# Преобразование входных текстов в последовательности индексов

# Применено приведение к массиву объектов для дальнейшего разделения на выборки

x\_data = np.array(tokenizer.texts\_to\_sequences(text\_data), dtype=object)

# Вывод формы и примера данных

print(x\_data.shape)

print(x\_data[0][:20])

# Преобразование входных текстов в разреженную матрицу из векторов Bag of Words

x\_data\_01 = tokenizer.texts\_to\_matrix(text\_data)

# Вывод формы и примера данных

print(x\_data\_01.shape)

print(x\_data\_01[0, :20])

# Преобразование меток класса к векторам one hot encoding

y\_data = utils.to\_categorical(class\_labels, CLASS\_COUNT)

# Вывод формы и примера данных

print(y\_data.shape)

print(y\_data[0])

# Получение индексов разделения основного набора на обучающую и тестовую выборки

idx\_train, idx\_test = train\_test\_split(list(range(len(x\_data))),

                                       stratify=y\_data,

                                       test\_size=TRAIN\_TEST\_RATIO)

# Разделение в соответствии с полученными индексами

x\_train, x\_test = x\_data[idx\_train], x\_data[idx\_test]

x\_train\_01, x\_test\_01 = x\_data\_01[idx\_train], x\_data\_01[idx\_test]

y\_train, y\_test = y\_data[idx\_train], y\_data[idx\_test]

# Проверка результата

print(x\_train.shape, x\_test.shape)

print(x\_train\_01.shape, x\_test\_01.shape)

print(y\_train.shape, y\_test.shape)

# Функция компиляции и обучения модели нейронной сети

def compile\_train\_model(model,

                        x\_train,

                        y\_train,

                        x\_val,

                        y\_val,

                        optimizer='adam',

                        epochs=50,

                        batch\_size=128,

                        figsize=(20, 5)):

    # Компиляция модели

    model.compile(optimizer=optimizer,

                  loss='categorical\_crossentropy',

                  metrics=['accuracy'])

    # Вывод сводки

    model.summary()

    # Обучение модели с заданными параметрами

    history = model.fit(x\_train,

                        y\_train,

                        epochs=epochs,

                        batch\_size=batch\_size,

                        validation\_data=(x\_val, y\_val))

    # Вывод графиков точности и ошибки

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)

    fig.suptitle('График процесса обучения модели')

    ax1.plot(history.history['accuracy'],

               label='Доля верных ответов на обучающем наборе')

    ax1.plot(history.history['val\_accuracy'],

               label='Доля верных ответов на проверочном наборе')

    ax1.xaxis.get\_major\_locator().set\_params(integer=True)

    ax1.set\_xlabel('Эпоха обучения')

    ax1.set\_ylabel('Доля верных ответов')

    ax1.legend()

    ax2.plot(history.history['loss'],

               label='Ошибка на обучающем наборе')

    ax2.plot(history.history['val\_loss'],

               label='Ошибка на проверочном наборе')

    ax2.xaxis.get\_major\_locator().set\_params(integer=True)

    ax2.set\_xlabel('Эпоха обучения')

    ax2.set\_ylabel('Ошибка')

    ax2.legend()

    plt.show()

# Функция вывода результатов оценки модели на заданных данных

def eval\_model(model, x, y\_true,

               class\_labels=[],

               cm\_round=3,

               title='',

               figsize=(10, 10)):

    # Вычисление предсказания сети

    y\_pred = model.predict(x)

    # Построение матрицы ошибок

    cm = confusion\_matrix(np.argmax(y\_true, axis=1),

                          np.argmax(y\_pred, axis=1),

                          normalize='true')

    # Округление значений матрицы ошибок

    cm = np.around(cm, cm\_round)

    # Отрисовка матрицы ошибок

    fig, ax = plt.subplots(figsize=figsize)

    ax.set\_title(f'Нейросеть {title}: матрица ошибок нормализованная', fontsize=18)

    disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=class\_labels)

    disp.plot(ax=ax)

    plt.gca().images[-1].colorbar.remove()  # Стирание ненужной цветовой шкалы

    plt.xlabel('Предсказанные классы', fontsize=16)

    plt.ylabel('Верные классы', fontsize=16)

    fig.autofmt\_xdate(rotation=45)          # Наклон меток горизонтальной оси при необходимости

    plt.show()

    print('-'\*100)

    print(f'Нейросеть: {title}')

    # Для каждого класса:

    for cls in range(len(class\_labels)):

        # Определяется индекс класса с максимальным значением предсказания (уверенности)

        cls\_pred = np.argmax(cm[cls])

        # Формируется сообщение о верности или неверности предсказания

        msg = 'ВЕРНО :-)' if cls\_pred == cls else 'НЕВЕРНО :-('

        # Выводится текстовая информация о предсказанном классе и значении уверенности

        print('Класс: {:<20} {:3.0f}% сеть отнесла к классу {:<20} - {}'.format(class\_labels[cls],

                                                                               100. \* cm[cls, cls\_pred],

                                                                               class\_labels[cls\_pred],

                                                                               msg))

    # Средняя точность распознавания определяется как среднее диагональных элементов матрицы ошибок

    print('\nСредняя точность распознавания: {:3.0f}%'.format(100. \* cm.diagonal().mean()))

# Совместная функция обучения и оценки модели нейронной сети

def compile\_train\_eval\_model(model,

                             x\_train,

                             y\_train,

                             x\_test,

                             y\_test,

                             class\_labels=CLASS\_LIST,

                             title='',

                             optimizer='adam',

                             epochs=50,

                             batch\_size=128,

                             graph\_size=(20, 5),

                             cm\_size=(10, 10)):

    # Компиляция и обучение модели на заданных параметрах

    # В качестве проверочных используются тестовые данные

    compile\_train\_model(model,

                        x\_train, y\_train,

                        x\_test, y\_test,

                        optimizer=optimizer,

                        epochs=epochs,

                        batch\_size=batch\_size,

                        figsize=graph\_size)

    # Вывод результатов оценки работы модели на тестовых данных

    eval\_model(model, x\_test, y\_test,

               class\_labels=class\_labels,

               title=title,

               figsize=cm\_size)

# Построение гистограммы распределения длин текстов основного набора

seq\_len = [len(x) for x in x\_data]

plt.hist(seq\_len, 50)

plt.show()

# Снижение размерности входных данных:

# ограничение длины последовательностей до разумного предела

seq\_max\_len = 400

x\_train\_clip = pad\_sequences(x\_train, maxlen=seq\_max\_len)

x\_test\_clip = pad\_sequences(x\_test, maxlen=seq\_max\_len)

# Проверка формы результата

print(x\_train\_clip.shape, x\_test\_clip.shape)

# Последовательная модель

model\_text\_emb\_dense = Sequential()

# Cлой эмбеддингов

model\_text\_emb\_dense.add(Embedding(VOCAB\_SIZE, 50, input\_length=seq\_max\_len))

# Слой регуляризации Dropout

model\_text\_emb\_dense.add(SpatialDropout1D(0.2))

# Cлой преобразования многомерных данных в одномерные

model\_text\_emb\_dense.add(Flatten())

# Слой пакетной нормализации

model\_text\_emb\_dense.add(BatchNormalization())

# Полносвязный слой

model\_text\_emb\_dense.add(Dense(64, activation="sigmoid"))

# Слой регуляризации Dropout

model\_text\_emb\_dense.add(Dropout(0.2))

# Слой пакетной нормализации

model\_text\_emb\_dense.add(BatchNormalization())

# Выходной полносвязный слой

model\_text\_emb\_dense.add(Dense(CLASS\_COUNT, activation='softmax'))

compile\_train\_eval\_model(model\_text\_emb\_dense,

                         x\_train\_clip, y\_train,

                         x\_test\_clip, y\_test,

                         epochs=20,

                         batch\_size=32,

                         cm\_size=(16, 16),

                         class\_labels=CLASS\_LIST,

                         title='Embedding + dense')

# Проверка сбалансированности выборок по классам

fig = plt.figure(figsize=(10, 5))

c\_train = np.bincount(np.argmax(y\_train, axis=1))

c\_test = np.bincount(np.argmax(y\_test, axis=1))

plt.bar(CLASS\_LIST, c\_train)

plt.bar(CLASS\_LIST, c\_test)

fig.autofmt\_xdate(rotation=45)

plt.show()

# Назначение интервала допустимого количества текстов по классу

text\_count\_interval = [100, 300]

# Фильтрация данных - отбрасываются все данные без категории

# и все классы объемом менее нижней границы разрешенного интервала

class\_to\_drop = ['нет данных']

for cls in df.category.unique():

    if df[df.category == cls].shape[0] < text\_count\_interval[0]:

        class\_to\_drop.append(cls)

print(f'Удаляются классы: {class\_to\_drop}')

del\_mask = df.category.isin(class\_to\_drop)

df = df[~del\_mask]

# По всем оставшимся классам количество примеров ограничивается не более

# верхней границы разрешенного интервала

for cls in df.category.unique():

    df = df.drop(df[df.category == cls].index[text\_count\_interval[1]:])

# Извелечение всех записей-текстов обращений и их меток классов

text\_data = df.text.tolist()

class\_data = df.category.tolist()

# Токенизация и построение частотного словаря по обучающим текстам

# Используется встроенный в Keras токенизатор для разбиения текста и построения частотного словаря

tokenizer = Tokenizer(num\_words=VOCAB\_SIZE, filters='!"#$%&()\*+,-–—./…:;<=>?@[\\]^\_`{|}~«»\t\n\xa0\ufeff', lower=True, split=' ', oov\_token='неизвестное\_слово', char\_level=False)

# Использованы параметры:

# num\_words   - объем словаря

# filters     - убираемые из текста ненужные символы

# lower       - приведение слов к нижнему регистру

# split       - разделитель слов

# char\_level  - указание разделять по словам, а не по единичным символам

# oov\_token   - токен для слов, которые не вошли в словарь

# Построение частотного словаря по обучающим текстам

tokenizer.fit\_on\_texts(text\_data)

# Построение словаря в виде пар слово - индекс

items = list(tokenizer.word\_index.items())

# Кодирование меток классов индексами (числами)

encoder = LabelEncoder()

class\_labels = encoder.fit\_transform(class\_data)

CLASS\_LIST = encoder.classes\_

CLASS\_COUNT = len(CLASS\_LIST)

print(f'Размер словаря: {len(items)}')

print(f'Список классов: {CLASS\_LIST}')

print(f'Всего классов: {CLASS\_COUNT}')

print(f'Форма выходных данных: {class\_labels.shape}')

print(f'Пример числовых меток классов: {class\_labels[:10]}')

# Преобразование входных текстов в последовательности индексов (для архитектур с embedding)

# Применено приведение к массиву объектов для дальнейшего разделения на выборки

x\_data = np.array(tokenizer.texts\_to\_sequences(text\_data), dtype=object)

# Вывод формы и примера данных

print(x\_data.shape)

print(x\_data[0][:20])

# Преобразование входных текстов в разреженную матрицу из векторов Bag of Words

x\_data\_01 = tokenizer.texts\_to\_matrix(text\_data)

# Вывод формы и примера данных

print(x\_data\_01.shape)

print(x\_data\_01[0, :20])

# Преобразование меток класса к векторам one hot encoding

y\_data = utils.to\_categorical(class\_labels, CLASS\_COUNT)

# Вывод формы и примера данных

print(y\_data.shape)

print(y\_data[0])

# Получение индексов разделения основного набора на обучающую и тестовую выборки

idx\_train, idx\_test = train\_test\_split(list(range(len(x\_data))),

                                       stratify=y\_data,

                                       test\_size=TRAIN\_TEST\_RATIO)

# Разделение в соответствии с полученными индексами

x\_train, x\_test = x\_data[idx\_train], x\_data[idx\_test]

x\_train\_01, x\_test\_01 = x\_data\_01[idx\_train], x\_data\_01[idx\_test]

y\_train, y\_test = y\_data[idx\_train], y\_data[idx\_test]

# Проверка результата

print(x\_train.shape, x\_test.shape)

print(x\_train\_01.shape, x\_test\_01.shape)

print(y\_train.shape, y\_test.shape)

# Проверка сбалансированности выборок по классам

fig = plt.figure(figsize=(10, 5))

c\_train = np.bincount(np.argmax(y\_train, axis=1))

c\_test = np.bincount(np.argmax(y\_test, axis=1))

plt.bar(CLASS\_LIST, c\_train)

plt.bar(CLASS\_LIST, c\_test)

fig.autofmt\_xdate(rotation=45)

plt.show()

# Последовательная модель

model\_text\_bow\_dense = Sequential()

# Входной полносвязный слой

model\_text\_bow\_dense.add(Dense(600, input\_dim=VOCAB\_SIZE, activation="sigmoid"))

# Слой регуляризации Dropout

model\_text\_bow\_dense.add(Dropout(0.25))

# Выходной полносвязный слой

model\_text\_bow\_dense.add(Dense(CLASS\_COUNT, activation='softmax'))

# Входные данные подаются в виде векторов bag of words

compile\_train\_eval\_model(model\_text\_bow\_dense,

                         x\_train\_01, y\_train,

                         x\_test\_01, y\_test,

                         epochs=20,

                         batch\_size=32,

                         cm\_size=(16, 16),

                         class\_labels=CLASS\_LIST,

                         title='BoW + dense')

# Кодирование меток классов индексами (числами)

encoder1 = LabelEncoder()

executor\_labels = encoder1.fit\_transform(executor\_data)

EXECUTOR\_LIST = encoder1.classes\_

EXECUTOR\_COUNT = len(EXECUTOR\_LIST)

print(f'Размер словаря: {len(items)}')

print(f'Список классов: {EXECUTOR\_LIST}')

print(f'Всего классов: {EXECUTOR\_COUNT}')

print(f'Форма выходных данных: {executor\_labels.shape}')

print(f'Пример числовых меток классов: {executor\_labels[:10]}')

# Преобразование входных текстов в последовательности индексов (для архитектур с embedding)

# Применено приведение к массиву объектов для дальнейшего разделения на выборки

x\_data = np.array(tokenizer.texts\_to\_sequences(text\_data), dtype=object)

# Вывод формы и примера данных

print(x\_data.shape)

print(x\_data[0][:20])

# Преобразование входных текстов в разреженную матрицу из векторов Bag of Words

x\_data\_01 = tokenizer.texts\_to\_matrix(text\_data)

# Вывод формы и примера данных

print(x\_data\_01.shape)

print(x\_data\_01[0, :20])

# Преобразование меток класса к векторам one hot encoding

y\_data = utils.to\_categorical(executor\_labels, EXECUTOR\_COUNT)

# Вывод формы и примера данных

print(y\_data.shape)

print(y\_data[0])

# Получение индексов разделения основного набора на обучающую и тестовую выборки

idx\_train, idx\_test = train\_test\_split(list(range(len(x\_data))),

                                       stratify=y\_data,

                                       test\_size=TRAIN\_TEST\_RATIO)

# Разделение в соответствии с полученными индексами

x\_train, x\_test = x\_data[idx\_train], x\_data[idx\_test]

x\_train\_01, x\_test\_01 = x\_data\_01[idx\_train], x\_data\_01[idx\_test]

y\_train, y\_test = y\_data[idx\_train], y\_data[idx\_test]

# Проверка результата

print(x\_train.shape, x\_test.shape)

print(x\_train\_01.shape, x\_test\_01.shape)

print(y\_train.shape, y\_test.shape)

# Функция компиляции и обучения модели нейронной сети

def compile\_train\_model1(model,

                        x\_train,

                        y\_train,

                        x\_val,

                        y\_val,

                        optimizer='adam',

                        epochs=50,

                        batch\_size=128,

                        figsize=(20, 5)):

    # Компиляция модели

    model.compile(optimizer=optimizer,

                  loss='categorical\_crossentropy',

                  metrics=['accuracy'])

    # Вывод сводки

    model.summary()

    # Обучение модели с заданными параметрами

    history = model.fit(x\_train,

                        y\_train,

                        epochs=epochs,

                        batch\_size=batch\_size,

                        validation\_data=(x\_val, y\_val))

    # Вывод графиков точности и ошибки

    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=figsize)

    fig.suptitle('График процесса обучения модели')

    ax1.plot(history.history['accuracy'],

               label='Доля верных ответов на обучающем наборе')

    ax1.plot(history.history['val\_accuracy'],

               label='Доля верных ответов на проверочном наборе')

    ax1.xaxis.get\_major\_locator().set\_params(integer=True)

    ax1.set\_xlabel('Эпоха обучения')

    ax1.set\_ylabel('Доля верных ответов')

    ax1.legend()

    ax2.plot(history.history['loss'],

               label='Ошибка на обучающем наборе')

    ax2.plot(history.history['val\_loss'],

               label='Ошибка на проверочном наборе')

    ax2.xaxis.get\_major\_locator().set\_params(integer=True)

    ax2.set\_xlabel('Эпоха обучения')

    ax2.set\_ylabel('Ошибка')

    ax2.legend()

    plt.show()

# Функция вывода результатов оценки модели на заданных данных

def eval\_model(model, x, y\_true,

               executor\_labels=[],

               cm\_round=3,

               title='',

               figsize=(10, 10)):

    # Вычисление предсказания сети

    y\_pred = model.predict(x)

    # Построение матрицы ошибок

    cm = confusion\_matrix(np.argmax(y\_true, axis=1),

                          np.argmax(y\_pred, axis=1),

                          normalize='true')

    # Округление значений матрицы ошибок

    cm = np.around(cm, cm\_round)

    # Отрисовка матрицы ошибок

    fig, ax = plt.subplots(figsize=figsize)

    ax.set\_title(f'Нейросеть {title}: матрица ошибок нормализованная', fontsize=18)

    disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=executor\_labels)

    disp.plot(ax=ax)

    plt.gca().images[-1].colorbar.remove()  # Стирание ненужной цветовой шкалы

    plt.xlabel('Предсказанные фамилии', fontsize=16)

    plt.ylabel('Верные фамилии', fontsize=16)

    fig.autofmt\_xdate(rotation=45)          # Наклон меток горизонтальной оси при необходимости

    plt.show()

    print('-'\*100)

    print(f'Нейросеть: {title}')

    # Для каждого класса:

    for cls in range(len(executor\_labels)):

        # Определяется индекс класса с максимальным значением предсказания (уверенности)

        cls\_pred = np.argmax(cm[cls])

        # Формируется сообщение о верности или неверности предсказания

        msg = 'ВЕРНО :-)' if cls\_pred == cls else 'НЕВЕРНО :-('

        # Выводится текстовая информация о предсказанном классе и значении уверенности

        print('Класс: {:<20} {:3.0f}% сеть отнесла к исполнителю {:<20} - {}'.format(executor\_labels[cls],

                                                                               100. \* cm[cls, cls\_pred],

                                                                               executor\_labels[cls\_pred],

                                                                               msg))

    # Средняя точность распознавания определяется как среднее диагональных элементов матрицы ошибок

    print('\nСредняя точность распознавания: {:3.0f}%'.format(100. \* cm.diagonal().mean()))

# Совместная функция обучения и оценки модели нейронной сети

def compile\_train\_eval\_model1(model,

                             x\_train,

                             y\_train,

                             x\_test,

                             y\_test,

                             executor\_labels=EXECUTOR\_LIST,

                             title='',

                             optimizer='adam',

                             epochs=50,

                             batch\_size=128,

                             graph\_size=(20, 5),

                             cm\_size=(10, 10)):

    # Компиляция и обучение модели на заданных параметрах

    # В качестве проверочных используются тестовые данные

    compile\_train\_model1(model,

                        x\_train, y\_train,

                        x\_test, y\_test,

                        optimizer=optimizer,

                        epochs=epochs,

                        batch\_size=batch\_size,

                        figsize=graph\_size)

    # Вывод результатов оценки работы модели на тестовых данных

    eval\_model(model, x\_test, y\_test,

               executor\_labels=executor\_labels,

               title=title,

               figsize=cm\_size)

# Последовательная модель

model\_text\_bow\_dense1 = Sequential()

# Входной полносвязный слой

model\_text\_bow\_dense1.add(Dense(100, input\_dim=VOCAB\_SIZE, activation="sigmoid"))

# Слой регуляризации Dropout

model\_text\_bow\_dense1.add(Dropout(0.4))

# Слой регуляризации Dropout

model\_text\_bow\_dense1.add(Dropout(0.4))

# Выходной полносвязный слой

model\_text\_bow\_dense1.add(Dense(EXECUTOR\_COUNT, activation='softmax'))

# Входные данные подаются в виде векторов bag of words

compile\_train\_eval\_model1(model\_text\_bow\_dense1,

                         x\_train\_01, y\_train,

                         x\_test\_01, y\_test,

                         epochs=20,

                         batch\_size=32,

                         cm\_size=(16, 16),

                         executor\_labels=EXECUTOR\_LIST,

                         title='BoW + dense')

!pip install pytelegrambotapi

import telebot

import PIL

from PIL import Image

from requests import get

#Сюда подставляем токен бота телеграмм

bot = telebot.TeleBot('--')

#Авторизация в бота

@bot.message\_handler(commands=['start'])

def start\_message(message):

    send = bot.send\_message(message.chat.id, 'Привет! Введите текст вашего Обращения')

    bot.register\_next\_step\_handler(send, get\_category)

#Логика бота

@bot.message\_handler(content\_types=['text'])

def send\_text(message):

   if message.text.lower() == 'создать обращение':

        send = bot.send\_message(message.chat.id, 'Введите текст вашего Обращения')

        bot.register\_next\_step\_handler(send, get\_category)

   else:

    bot.send\_message(message.chat.id, 'Напишите "создать обращение"')

#Использование предсказаний бота на основе набранного текста

def get\_category(message):

  class1 = tokenizer.texts\_to\_matrix([message.text])

  classification\_class = model\_text\_bow\_dense.predict(class1)

  execution1 = tokenizer.texts\_to\_matrix([message.text])

  classification\_execution = model\_text\_bow\_dense1.predict(execution1)

  print(classification\_class)

  print(classification\_execution)

  clas\_pred = np.argmax(classification\_class)

  print(clas\_pred)

  exec\_pred = np.argmax(classification\_execution)

  df['category'].unique()[clas\_pred-1]

  bot.send\_message(message.chat.id, 'Ваше обращение зарегистрировано. Категория данного обращения - {0}. Исполнитель - {1}'.format(df['category'].unique()[clas\_pred-1], df['executor'].unique()[exec\_pred-1]))

bot.polling()