

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**

Высшая школа электроники и компьютерных наук

Кафедра системного программирования

РАБОТА ПРОВЕРЕНА

Рецензент

Руководитель отдела машинного
обучения компании «Интерсвязь»

_____ Ю.В. Дмитрин

«___» _____ 2021 г.

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н.,
профессор

_____ Л.Б. Соколинский

«___» _____ 2021 г.

**Разработка интеллектуальной системы автоматизированной
обработки обращений граждан**

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 02.04.02.2021.308-440.ВКР

Научный руководитель,
профессор кафедры СП, д.ф.-м.н.,
доцент

_____ Р.Ж. Алеев

Автор работы,
студент группы КЭ-220

_____ А.В. Витомсков

Ученый секретарь
(нормоконтролер)

_____ И.Д. Володченко

«___» _____ 2021 г.

Челябинск, 2021 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования

**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**

**Высшая школа электроники и компьютерных наук
Кафедра системного программирования**

УТВЕРЖДАЮ

Зав. кафедрой СП

_____ Л.Б. Соколинский

08.02.2021 г.

ЗАДАНИЕ

**на выполнение выпускной квалификационной работы магистра
студенту группы КЭ-220**

**Витомскову Алексею Вадимовичу,
обучающемуся по направлению**

**02.04.02 «Фундаментальная информатика и информационные технологии»
(магистерская программа «Машинное обучение и анализ больших данных»)**

1. Тема работы (утверждена приказом ректора от 26.04.2021 г. № 714-13/12)

Разработка интеллектуальной системы автоматизированной обработки
обращений граждан.

2. Срок сдачи студентом законченной работы: 28.05.2021 г.

3. Исходные данные к работе

3.1. Открытый набор данных обращений граждан региона Татарстан.

[Электронный ресурс] URL: <https://git.asi.ru/tasks/world-ai-and-data-challenge/intelligent-algorithm-for-automated-processing-of-citizenss-requests> (дата обращения: 13.05.2021 г.).

4. Перечень подлежащих разработке вопросов

4.1. Поиск и анализ существующих решений.

4.2. Предобработка данных.

4.3. Исследование различных векторных моделей для работы с текстом:
статистических и нейросетевых.

4.4. Выбор метрики качества.

- 4.5. Выбор схемы валидации, оценка устойчивости модели.
- 4.6. Сравнение результатов моделей, анализ ошибок моделей.
- 4.7. Разработка приложения, реализующего API для работы с моделью.
- 4.8. Тестирование разработанного приложения.

5. Дата выдачи задания: 08.02.2021.

Научный руководитель,
профессор кафедры СП, д.ф.-м.н., доцент

Р.Ж. Алеев

Задание принял к исполнению

А.В. Витомсков

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ.....	5
1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ	8
1.1. Обзор существующих решений.....	8
1.2. Обзор программных аналогов	9
2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	10
2.1. Обзор алгоритмов предобработки текстовых данных	11
2.2. Обзор статистических моделей обработки естественного языка	17
2.3. Обзор нейросетевых моделей обработки естественного языка ..	25
2.4. Метрики качества моделей	31
3. ПРОЕКТИРОВАНИЕ	34
3.1. Требования к системе	34
3.2. Варианты использования системы.....	34
3.3. Проектирование архитектуры системы.....	36
3.4. Проектирование серверной части	37
3.5. Проектирование микросервиса на архитектуре REST.....	39
3.6. Проектирование интерфейса пользователя.....	40
4. РЕАЛИЗАЦИЯ	42
4.1. Средства реализации	42
4.2. Подготовка данных для обучения.....	43
4.3. Реализация алгоритмов анализа текстовых данных.....	45
4.4. Реализация микросервиса на архитектуре REST.....	59
4.5. Реализация интерфейса пользователя.....	62
5. ТЕСТИРОВАНИЕ	68
ЗАКЛЮЧЕНИЕ	72
ЛИТЕРАТУРА.....	73

ВВЕДЕНИЕ

Актуальность темы исследования

В настоящее время цифровая трансформация в социальной сфере стала ключевым трендом. Упрощается система коммуникации между поставщиками услуг и потребителями, растет количество обращений граждан в социальные службы.

Государственный сектор – одна из главных площадок реализации правительственных инициатив в сфере ИТ. Гражданин может обратиться в госучреждение устно, по почте, с помощью телефонного звонка, электронного письма или оформления обращения через веб-сайт учреждения. Обращение должно быть рассмотрено в сжатые сроки и с гарантированным качеством вне зависимости от формы обращения.

С развитием информационной культуры граждан растет число обращений, следовательно, высока вероятность увеличения времени на их рассмотрение и формирование решений.

Сегодня в работе органов государственной власти и местного самоуправления в большинстве случаев встречаются элементы интерактивной работы с гражданами, однако зачастую имеющиеся технические решения слабо готовы к реализации полного цикла работы с обращениями граждан.

В начале января 2021 года президент России Владимир Путин поручил до 1 мая принять законы о масштабном применении искусственного интеллекта в стране. Речь идет о федеральных законах, по которым можно будет вводить экспериментальные режимы в отдельных отраслях для расширения использования ИИ.

Так, в рамках цифровой трансформации Росстата будет внедрено решение, обеспечивающее автоматическое распознавание и классификацию обращений граждан, поступающих в систему электронного документооборота Росстата. В рамках цифровой трансформации Роспотребнадзора будет осуществлено совершенствование сервиса ведомства по работе с обращени-

ями граждан путем внедрения технологий ИИ для автоматизации предварительных консультаций («онлайн-чат-бот»), процесса обработки обращений (онлайн и офлайн) и подготовки ответов.

Все это подтверждает актуальность затронутой темы.

Исходные данные

Исходными данными к работе служит открытый набор данных обращений граждан региона Татарстан. В задании имеется ссылка на этот набор данных. Есть аналогичные данные и от других регионов, но они попадают под NDA (соглашение о неразглашении конфиденциальной информации) компании «Интерсвязь» и не были использованы в данной работе.

За 2020 год только в Министерство образования и науки Республики Татарстан поступило 7656 обращений (в том числе 7570 в письменном виде). Основными источниками поступления обращений являются следующие: интернет-приемная, электронная почта, почта, электронный документооборот. При этом наиболее распространенная тематика обращений – социальная сфера и жилищно-коммунальная сфера [1].

Набор данных республики Татарстан содержит 31312 записей - текстов обращений граждан, которым соответствуют 37 тематических меток категорий и 121 метка об исполнителе.

Набор данных включает следующие поля:

- 1) дата подачи заявки;
- 2) категория, присвоенная модератором;
- 3) последний исполнитель;
- 4) ID исполнителя;
- 5) статус заявки.

Цель и задачи исследования

Целью работы является разработка интеллектуальной системы автоматизированной обработки обращений граждан, обеспечивающей их эффективную классификацию. Для достижения этой цели необходимо решить следующие задачи:

- 1) провести анализ существующих решений и методов автоматизированной обработки обращений граждан;
- 2) спроектировать архитектуру системы;
- 3) выбрать и обучить модели для работы с текстовыми данными с предварительной подготовкой данных;
- 4) провести тестирование моделей, отобрать лучшие и сравнить результаты с существующими решениями;
- 5) разработать веб-интерфейс для интеллектуальной системы автоматизированной обработки обращений граждан;
- 6) провести тестирование реализованной системы.

Структура и объем работы

Выпускная квалификационная работа состоит из введения, пяти основных разделов, заключения и библиографии. Объем работы составляет 75 страниц, объем библиографии – 23 наименования.

Краткий обзор содержания работы

В первом разделе «Анализ предметной области» приводится обзор существующих решений и программных аналогов. Во втором разделе «Теоретическая часть» описывается обзор алгоритмов предобработки текстовых данных, статистических и нейросетевых векторных моделей для обработки естественного языка, а также описываются метрики качества. В третьем разделе «Проектирование» описываются требования к системе, варианты использования системы, приводится проектирование архитектуры в целом, а также ее компонентов, таких как серверная часть и веб-интерфейс, а также проектирование алгоритмов анализа текстовых данных. В четвертом разделе «Реализация» описываются средства реализации, как осуществлялась подготовка данных для обучения, а также реализация серверной части, веб-интерфейса и алгоритмов анализа текстовых данных. В пятом разделе «Тестирование» приводятся результаты тестирования веб-интерфейса и алгоритмов анализа текстовых данных. В заключении приводятся основные результаты работы и направления дальнейших исследований.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ

1.1. Обзор существующих решений

В области автоматизированной обработки обращений граждан на настоящее время проводилось большое количество исследований. Однако большая часть этих исследований основано на эмпирическом подходе и нацелены на работу с обращениями весьма узкого профиля.

Так, в работе [2] Антонов С.В. проводит исследование влияния способа представления обращений о ЧС (чрезвычайных ситуациях) на скорость их обработки «Системой 112». Телефонный звонок оказался самым медленным способом обработки сообщений. Обработка диспетчером текстовых сообщений выполнялась намного быстрее, а обработка предварительно обработанных и классифицированных по ключевым словам сообщений показала максимальный результат. Для классификации автором статьи использовалось 120 ключевых слов. Главной задачей, поставленной автором, была задача нахождения ключевых слов.

В настоящее время мы наблюдаем стремительное развитие в области искусственного интеллекта. Накоплены большие объемы тренировочных данных; разработаны вычислительные мощности: многоядерные CPU и GPU; созданы новые модели и алгоритмы с расширенными возможностями и улучшенной производительностью, с гибким обучением на промежуточных представлениях. Появились обучающие методы с использованием контекста, новые методы регуляризации и оптимизации.

В то же время современных систем классификации обращений на основе алгоритмов машинного обучения, которые могут работать сразу во многих специализированных областях знаний, разработано мало. И особенно это касается моделей для работы с русским языком. Это прибавляет актуальности моему исследованию.

1.2. Обзор программных аналогов

На данный момент на рынке существуют программные аналоги интеллектуальной системы обработки обращений граждан.

Так, командой специалистов из г. Рязань разработан классификатор обращений граждан [3]. В данном классификаторе выводятся топ 3 возможных варианта темы обращения, категории обращения и исполнителей с сортировкой в порядке убывания вероятности совпадения.

Авторы работы используют технологии Word2Vec и Word Mover's Distance (WMD), описанные в работах [4], [5]. Данный подход позволяет создать единое семантическое пространство для представления отдельных слов. Расстояние между двумя текстовыми документами А и В в таком случае можно рассчитать по минимальному совокупному расстоянию, которое должны пройти слова из текстового документа А, чтобы точно соответствовать облаку точек текстового документа В. Для каждого нового документа рассчитывается вектор расстояний от данного документа до каждого документа из обучающего набора. Затем применяется классификатор k-ближайших соседей с параметром количества соседей, равным 10-и.

В результате данного исследования, авторами были получены значения метрики ассигасу, представленные в таблице 1.

Таблица 1 – Метрики ассигасу качества классификатора

Условие – попадание в Топ-3	Категории	Исполнители	Темы
1-е место	1076 (60.2%)	1127 (63.0%)	792 (44.3%)
2-е и 3-е место	411 (23.0%)	357 (20.0%)	80 (15.7%)
Не попало в Топ-3	301 (16.8%)	304 (17.0%)	716 (40.0%)

Командой специалистов из г. Пермь разработан автоматический классификатор обращений граждан [6]. В качестве модели машинного обучения авторами использовался метод опорных векторов. Строились отдельные модели для определения категории, темы и исполнителя. Поступающие для анализа новые обращения граждан использовались авторами для дообучения существующих моделей.

2. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Обработка естественного языка (Natural Language Processing, NLP) – область исследований на границе машинного обучения и математической лингвистики, направленных на изучение методов анализа и синтеза естественного языка. Сегодня NLP применяется во многих сферах, в том числе в голосовых помощниках, автоматических переводах текста и фильтрации текста. Основными тремя направлениями являются: распознавание речи (Speech Recognition), понимание естественного языка (Natural Language Understanding) и генерация естественного языка (Natural Language Generation).

В данной работе были рассмотрены область понимания естественного языка и связанные с ней статистические и нейросетевые модели.

Система автоматизированной обработки обращений граждан позволит проводить анализ и обработку в двух режимах:

- 1) анализ отдельного обращения – определение для поступившего обращения:
 - тематики обращения (топ-3 вероятных класса);
 - темы обращения (на основе неразмеченных данных);
 - ключевых слов и именованных сущностей;
 - предполагаемого типа исполнителя;
- 2) обзор обращений граждан.

Для работы интеллектуальной системы в первом режиме необходима разработка как статистических, так и нейросетевых моделей обработки естественного языка.

Для определения тематики обращений мной были исследованы различные топологии нейросетей, в итоге применена нейросетевая модель BERT на основе механизма внимания.

С целью сузить область поиска исполнителя, в данной работе также в рамках каждой категории будет находиться от 3 до 10 тем. С этой задачей

хорошо справляются иерархические тематические модели на основе аддитивной регуляризации. Такие модели представляют собой модели обучения без учителя. Данная модель также используется для определения ключевых слов.

Для нахождения именованных сущностей из текстов обращений (названий организаций, локаций, персон, дат и т. д.) были исследованы нейросетевые NER-модели.

2.1. Обзор алгоритмов предобработки текстовых данных

Предварительная обработка данных заключается в получении этих данных в форму, которую в дальнейшем можно использовать для обучения.

Перед построением статистических, либо нейросетевых моделей текст естественного языка обычно подвергается серии преобразований.

Форматирование, очистка, отбор проб

Для начала данные необходимо очистить. Очистка данных предполагает удаление данных, которые являются неполными и не содержат данных, которые необходимы для решения задачи. Так, были отфильтрованы некоторые низкочастотные классы с числом примеров в наборе данных менее 10. Кроме того, были удалены короткие обращения, содержащие менее 3-х слов. В итоге количество категорий составило 26.

Так как исходный набор данных содержит не слишком большое количество данных, поэтому отбор проб не применялся, и все дальнейшие операции выполнялись на полном наборе данных.

Нормализация

Нормализация текстовых данных обычно состоит из таких операций, как приведение всех символов к нижнему регистру, исправление опечаток, удаление цифр и спецсимволов, упорядочивание формы слов (стемминг или лемматизация [7]), удаление стоп-слов, удаление редких слов, выделение ключевых фраз и распознавание именованных сущностей. Нормализация

имеет целью сократить объем словаря, улучшить результаты решения и сократить время решения задачи.

С целью упорядочивания формы слова существуют два подхода: стемминг и лемматизация.

Стемминг – это отбрасывание окончаний и других изменяемых частей слов. Он хорошо подходит для английского языка, для русского же предпочтительна лемматизация.

Лемматизация – это процесс приведения слова к его словарной форме. При лемматизации используется знание о точной части речи слова, поэтому можно с большей уверенностью предположить, как будет выглядеть его начальная форма.

Стоп-слова – это частые слова, встречающиеся в текстах любой тематики – предлоги, союзы, числительные, местоимения. Отбрасывание таких слов почти не влияет на объем словаря, но может приводить к заметному сокращению длины текстов.

Удаление редких слов и строк, не являющихся словами естественного языка, помогает во много раз сокращать объем словаря, снижая затраты времени и памяти на построение моделей. Редкие слова, как правило, не влияют на тематику коллекции.

Выделение ключевых фраз используется для улучшения интерпретируемости тем. Выделять их можно с помощью тезаурусов или методов автоматического выделения терминов (automatic term extraction, АТЕ), работающих по принципу обучения без учителя.

Также на этапе нормализации может потребоваться нахождение именованных сущностей – названий объектов реального мира. Эти объекты могут относиться к определенным категориям: персоны, организации, геолокации, события, даты и т.д. Известны методы распознавания именованных сущностей (named entities recognition, NER), основанные на моделях машинного обучения.

Построение векторной модели

Большинство алгоритмов машинного обучения оперируют вещественно-пространственными признаками документов, или иными словами – векторной моделью. Векторная модель – это математическая модель представления текстов, в которой каждому документу $d \in D$, где D – некоторое множество документов, сопоставлен вектор слов, состоящий из всех возможных слов коллекции.

Классический случай – это построение разреженных векторных моделей. Такие модели еще называют «методом мешка слов» (Bag of Words). Они хорошо работают, когда класс документа соответствует его тематике. Как правило, тематика документа хорошо описывается составом словаря, который используется в этом документе, а также частотами слов, а не тем, как именно они употребляются в документе, не структурой фраз. Аналогичные методы, применяемые для N-грамм, называются Bag of N-grams.

Разреженные векторные модели должны работать лучше, когда веса слов отличаются – чем значимее слово, тем больше его вес. Простейший вариант – взвешивать слова по количеству их употреблений в документе. Но данный подход имеет недостатки. Во-первых, вес слова зависит от длины документа. В длинных документах слова имеют больший вес, как будто бы они более значимы, но это не так. Во-вторых, самые частотные слова – это союзы, предлоги, местоимения, не несущие важной для нас информации.

Чтобы избавиться от этих недостатков, вначале нужно пронормировать вектор документа, поделив на его длину по L2-норме (или по евклидовой норме) по формуле (1):

$$nw_i = \frac{w_i}{\sqrt{\sum_j w_j^2}}, \quad (1)$$

где nw_i – нормированное количество употреблений слова в документе.

На рисунке 1 представлен график распределения Ципфа, применимый для естественного языка. По оси ординат отложена относительная частота слова, а по оси абсцисс – ранг слова (его порядковый номер).

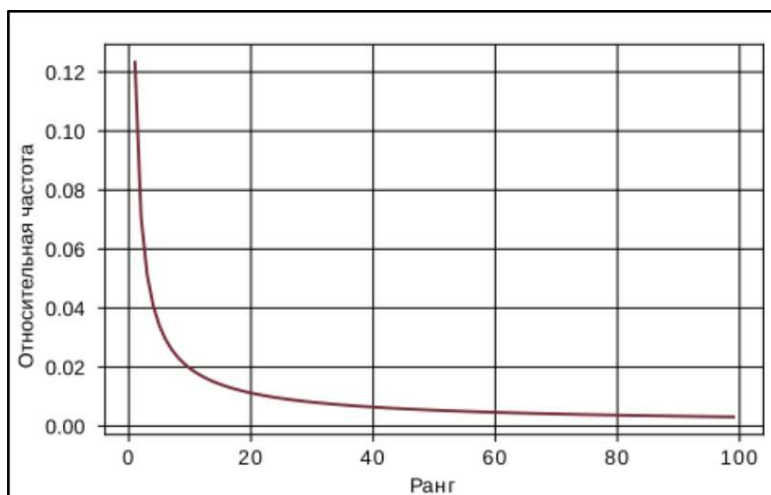


Рисунок 1 – График распределения Ципфа

Распределение Ципфа относится к классу степенных. Из его графика можно сделать два практических вывода: во-первых, частотных слов очень мало, и они слабо информативны. А во-вторых, редких слов очень много – если мы какое-нибудь редкое слово встречаем в документе, то мы с большой уверенностью можем сказать, к какой тематике он относится. Но проблема в том, что такие слова очень редки, и поэтому ненадежны в качестве факторов при принятии решений. Следовательно, нам нужно придерживаться баланса частотности и информативности. За этот баланс отвечают две величины: TF и IDF [8].

TF (term frequency) – это частота слова в документе, которая рассчитывается по формуле (2):

$$TF(w, d) = \frac{WordCount(w, d)}{Length(d)}, \quad (2)$$

где $WordCount(w, d)$ – количество употреблений слова w в документе d ; $Length(d)$ – длина документа d в словах.

IDF (inverse document frequency) – обратная частота слова в документах (специфичность слова), которая рассчитывается по формуле (3):

$$IDF(w, c) = \frac{Size(c)}{DocCount(w, c)}, \quad (3)$$

где $DocCount(w, c)$ – количество документов в коллекции c , в которых встречается слово w ;

$Size(c)$ – размер коллекции c в документах.

В итоге получаем формулу статистической меры TF-IDF (4):

$$TFIDF(w, d, c) = TF(w, d) \cdot IDF(w, c). \quad (4)$$

TF-IDF – это неплохой способ взвешивания и отбора категориальных признаков в задачах машинного обучения. Но существуют и другие способы взвешивания признаков по частоте – например, взаимная информация. Она измеряется между двумя случайными событиями или реализациями двух случайных величин. Она характеризует, насколько сильнее мы будем ожидать первое событие, если перед этим пронаблюдаем второе (по сравнению с нашими априорными ожиданиями).

Алгоритмы токенизации

При анализе текстовые данные не всегда разбивают на отдельные слова. Более того, есть языки, для которых это сделать в принципе невозможно. Поэтому, в более общем случае, текст разбивается на токены.

Токены – это символы, слова или N-граммы (комбинации слов). Способ представления токенов будет существенно влиять на качество обучаемой модели.

Классический подход подразумевает представление текста в виде символов, слов или N-грамм. Более продвинутые токенизаторы используют токенизацию пробелов и знаков препинания, а также токенизацию на основе правил. Однако, такой метод токенизации может привести к проблемам с массивными текстовыми корпусами. В этом случае использование пробелов и знаков препинания обычно создает очень большой словарный запас.

С другой стороны, токенизация отдельных символов очень проста и способна значительно снизить объем памяти и временную сложность. Но такая модель значительно усложняет изучение осмысленных представлений

ввода. Выучить осмысленное независимое от контекста представление для одной буквы намного сложнее, чем выучить независимое от контекста представление для слова. Таким образом, чтобы взять лучшее от этих двух идей, мы приходим к рассмотрению токенизаторов, использующих токенизацию подслова.

Алгоритмы разметки подслов основаны на принципе, согласно которому часто используемые слова не следует разбивать на более мелкие подслова, а редкие слова следует разбивать на значимые подслова. Это особенно полезно в агглютинативных языках, таких как турецкий, где можно образовывать (почти) произвольно длинные сложные слова, соединяя подслова вместе.

Токенизация подслов позволяет модели иметь разумный объем словарного запаса, в то же время имея возможность изучать значимые контекстно-независимые представления. Кроме того, токенизация подслов позволяет модели обрабатывать слова, которые она никогда раньше не видела, путем разложения их на известные подслова.

Так, токенизаторы на основе подслов применяются в трансформерах, которые используют три основных типа токенизаторов: Byte-Pair Encoding (BPE), WordPiece и Unigram.

Кодирование пар байтов (BPE) [9] – это простой метод, который итеративно заменяет наиболее частую пару байтов в последовательности одним неиспользуемым байтом. Этот алгоритм можно адаптировать для сегментации слов. Вместо того, чтобы объединять частые пары байтов, мы объединяем символы или последовательности символов. Размер словаря символов в таком случае равен размеру исходного словаря плюс количество операций слияния. Токенизатор BPE используют такие нейросетевые модели, как GPT-2 и Roberta.

WordPiece – это алгоритм токенизации подслов, используемый для таких трансформеров, как BERT, DistilBERT и Electra. Алгоритм был описан в статье о японском и корейском голосовом поиске [10] и очень похож на

BPE. WordPiece сначала инициализирует словарь, чтобы включить каждый символ, присутствующий в обучающих данных, и постепенно изучает заданное количество правил слияния. В отличие от BPE, WordPiece выбирает не самую частую пару символов, а ту, которая максимизирует вероятность того, что обучающие данные будут добавлены в словарь. Таким образом, WordPiece можно считать более эффективной реализацией BPE.

Unigram – это алгоритм токенизации подслов, представленный в статье [11]. В отличие от BPE или WordPiece, Unigram инициализирует свой базовый словарь большим количеством символов и постепенно сокращает каждый символ, чтобы получить меньший словарный запас. Базовый словарь может, например, соответствовать всем предварительно токенизированным словам и наиболее распространенным подстрокам.

2.2. Обзор статистических моделей обработки естественного языка

В данной работе мной были исследованы следующие статистические модели обработки естественного языка применительно к задаче многоклассовой классификации: логистическая регрессия (logistic regression), деревья решений (decision trees), ансамбли деревьев решений (decision tree ensembles) и метод опорных векторов (SVM, support vector machine). Наконец, будут рассмотрены иерархические тематические модели на основе аддитивной регуляризации.

Логистическая регрессия

Логистическая регрессия – это статистический метод, который используется для прогнозирования вероятности исхода, и особенно популярен для задач классификации. В отличие от обычной регрессии, в методе логистической регрессии не производится предсказание значения числовой переменной исходя из выборки исходных значений. Вместо этого, значением функции является вероятность того, что данное исходное значение принадлежит к определенному классу.

Основная идея логистической регрессии заключается в том, что пространство исходных значений может быть разделено границей решений (decision boundary) на две соответствующих классам области. Затем решение данной задачи можно распространить на множество классов.

Если рассматривать функцию границы решений $h(x)$ как оценку вероятности события $y = 1$ для данного x , то в качестве $h(x)$ удобно использовать нелинейную логистическую функцию сигмоиды (5):

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}, \quad (5)$$

где θ – некоторая матрица коэффициентов, которые нам нужно подобрать.

Применяя метод максимального правдоподобия, можно найти целевую функцию в задаче логистической регрессии. Она носит название «логлосс» и определяется формулой (6):

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m y^{(i)} \ln(h(x^{(i)})) + (1 - y^{(i)}) \ln(1 - h(x^{(i)})), \quad (6)$$

где $y^{(i)}$ – метка i -го обучающего образца;

$x^{(i)}$ – i -й обучающий пример;

m – количество образцов для обучения.

Использование логлосса в качестве целевой функции дает задачу выпуклой оптимизации. Производную от функции логлосса по θ можно найти по формуле (7):

$$\frac{\partial}{\partial \theta} J(\theta) = \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) x_j^{(i)}. \quad (7)$$

Подставив производную логлосса по θ в алгоритм градиентного спуска, получим формулу градиентного спуска (8):

$$\theta_j^{new} = \theta_j^{old} - \alpha \frac{1}{m} \sum_{i=1}^m (h(x^{(i)}) - y^{(i)}) x_j^{(i)}. \quad (8)$$

Решение задачи двухклассовой классификации теперь можно распространить на задачу множественной классификации, применив алгоритм

«один-против-всех» (one-vs-all). Построим n бинарных классификаторов (n – количество классов) (9):

$$h^{(i)}(x) = P\{y = i; x, \theta\}, \quad (9)$$

где $i = 1, \dots, n$.

Для данного x выберем класс с максимальным значением: $\max_i h^{(i)}(x)$.

Применительно к рассматриваемой задаче классификации текстовых данных, количество параметров θ равно размеру словаря. Предварительную векторизацию данных можно выполнить, например, при помощи TF-IDF.

Деревья решений

Деревья решений – это очень распространенный и эффективный метод классификации и регрессии, он относится к контролируемому обучению и имеет древовидную структуру. Каждый внутренний узел представляет тест по атрибуту, каждая ветвь представляет результаты теста, а каждый конечный узел представляет собой пример, это индуктивное обучение на основе примеров.

Несмотря на свою интерпретируемость и высокую выразительную способность, деревья крайне трудны для оптимизации из-за своей дискретной структуры – дерево нельзя продифференцировать по параметрам и найти хотя бы локальный минимум с помощью градиентного спуска. Базовый жадный алгоритм построения бинарного решающего дерева выглядит следующим образом. Начнем со всей обучающей выборки X_m и найдем наилучшее ее разбиение на две части с точки зрения заранее заданного критерия ошибки $Q(X, j, t)$ по формулам (10), (11):

$$X_l = \{x \in X_m \mid [x_j \leq t]\}, \quad (10)$$

$$X_r = \{x \in X_m \mid [x_j > t]\}. \quad (11)$$

Найдя наилучшие значения j и t , создадим корневую вершину дерева, поставив ей в соответствие предикат $[x_j \leq t]$. Объекты разобьются на две части – одни попадут в левое поддерево, другие в правое. Для каждой из

этих подвыборок рекурсивно повторим процедуру, построив дочерние вершины для корневой, и так далее. В каждой вершине мы проверяем, не выполнилось ли некоторое условие останова и если выполнилось, то прекращаем рекурсию и объявляем эту вершину листом.

На практике выбор наилучшего из всех возможных деревьев решений является NP-полной задачей, поэтому мы получаем субоптимальное дерево.

Рассмотрим критерий ошибки $Q(X, j, t)$, с помощью которого можно выбирать оптимальное разбиение при построении решающего дерева.

Критерий ошибки можно формально представить в виде (12):

$$Q(X_m, j, t) = \frac{|X_l|}{|X_m|} H(X_l) + \frac{|X_r|}{|X_m|} H(X_r), \quad (12)$$

где функция $H(X)$ называется критерием информативности.

$H(X)$ должна измерить, насколько силен разброс ответов в множестве X . Ее значение должно быть тем меньше, чем меньше разброс этих ответов. Таким образом, для получения более оптимального разбиения мы должны минимизировать Q .

Применительно к задаче классификации, можно ввести дополнительную величину p_k для k -го класса, которая показывает, какова доля объектов класса k в выборке X , и выражается формулой (13):

$$p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k]. \quad (13)$$

Один из критериев информативности для классификации – это критерий Джини (14):

$$H(X) = \sum_{k=1}^K p_k (1 - p_k). \quad (14)$$

Из этой формулы следует, что если все объекты выборки X относятся к одному классу, то критерий Джини равен нулю, иначе он больше нуля. Чем больше критерий Джини, тем выше установленная неопределенность и тем больше примесей.

Еще один критерий информативности для классификации – это энтропийный критерий (15):

$$H(X) = - \sum_{k=1}^K p_k \ln(p_k). \quad (15)$$

Для этого критерия выполнено то же самое свойство: если все объекты выборки X относятся к одному классу, то энтропийный критерий равен нулю, иначе он больше нуля. Если распределение выборок по классам равномерное, то энтропийный критерий максимален.

Стратегии, позволяющие предотвратить переобучение деревьев решений:

- 1) ранняя остановка построения дерева, pre-pruning (предварительная обрезка) – ограничение максимальной глубины дерева, максимального количества листьев;
- 2) построение дерева с последующим удалением или сокращением малоинформативных узлов, pruning (пост-обрезка).

Ансамбли деревьев решений

Ансамбли деревьев решений (decision tree ensembles) – это методы, которые сочетают в себе множество моделей машинного обучения, чтобы в итоге получить более мощную модель.

Существует две ансамблевых модели, которые доказали свою эффективность на самых различных наборах данных для задач классификации и регрессии, обе используют деревья решений в качестве строительных блоков:

- 1) случайный лес деревьев решений;
- 2) градиентный бустинг деревьев решений.

Случайный лес – это набор деревьев решений, где каждое дерево немного отличается от остальных. Каждое дерево такого набора может довольно хорошо прогнозировать, но скорее всего переобучается на части дан-

ных. Если мы построим много деревьев, которые хорошо работают и переобучаются с разной степенью, мы можем уменьшить переобучение путем усреднения их результатов.

В отличие от случайного леса, градиентный бустинг строит последовательность деревьев, в которой каждое дерево пытается исправить ошибки предыдущего.

Первую модель мы обучаем на случайной выборке из обучающего набора данных. Далее мы тестируем обученную модель на всех данных обучающего набора и отмечаем, на каких объектах наша модель ошибается больше всего.

Далее мы строим новый набор данных, вновь случайно выбирая образцы из обучающей выборки, но в этот раз каждый из них имеет определенный вес, зависящий от ошибки первой обученной модели на нем. Образцы, которые доставляют большую ошибку модели, имеют больше шансов попасть в новый набор данных. На этих данных мы обучаем вторую модель. Далее таким же образом мы строим новые наборы данных и новые модели, исправляющие ошибки существующего ансамбля.

Помимо предварительной обрезки и числа деревьев в ансамбле, еще один важный параметр градиентного бустинга – это `learning_rate`, который контролирует, насколько сильно каждое дерево будет пытаться исправить ошибки предыдущих деревьев. Более высокая скорость обучения означает, что каждое дерево может внести более сильные корректировки и это позволяет получить более сложную модель.

Тематические модели на основе аддитивной регуляризации

Для поиска тем в рамках найденных категорий будут использованы вероятностные тематические модели. Тема – это условное распределение на множестве терминов, $p(w|t)$ – вероятность (частота) термина w в теме t . Тематическая модель автоматически выявляет латентные темы по наблюдаемым частотам терминов в документах $p(w|d)$.

Сделаем базовые предположения:

- 1) порядок документов в коллекции не важен;
- 2) порядок терминов в документе не важен (bag of words);
- 3) употребление каждого слова в каждом документе (d, w) связано с некоторой темой $t \in T$;
- 4) слова в документе генерируются именно темой, а не самим документом.

Тогда условную вероятность появления слов в документе можно записать в виде (16):

$$p(w|d) = \sum_{t \in T} \varphi_{wt} \theta_{td}, \quad (16)$$

где $\varphi_{wt} = p(w|t)$ – вероятности терминов w в каждой теме t ;

$\theta_{td} = p(t|d)$ – вероятности тем t в каждом документе d .

Наблюдаемая частота слов в документе описывается как (17):

$$\hat{p}(w|d) = \frac{n_{dw}}{n_d}, \quad (17)$$

где n_{dw} – количество повторений слова w в документе d ;

n_d – количество слов в документе d .

Решаем задачу: по наблюдаемой коллекции, необходимо понять, какими распределениями φ_{wt} и θ_{td} она могла бы быть получена.

Формула (16) представляет тематическую модель как матричное разложение. Задача матричного разложения некорректно поставлена, поскольку ее решение в общем случае не единственно. Поэтому мы можем вводить дополнительные ограничения на матрицы Φ и Θ . Чтобы из множества решений выбрать наиболее подходящее, вводится критерий регуляризации $R(\Phi, \Theta)$.

Используя принцип максимума правдоподобия, получаем оптимизационную задачу (18):

$$\left\{ \begin{array}{l} \sum_{d \in D} \sum_{w \in W} n_{dw} \ln \sum_{t \in T} \varphi_{wt} \theta_{td} + R(\Phi, \Theta) \rightarrow \max_{\Phi, \Theta} \\ \sum_{w \in W} \varphi_{wt} = 1, \quad \varphi_{wt} \geq 0 \\ \sum_{t \in T} \theta_{td} = 1, \quad \theta_{td} \geq 0 \end{array} \right. \quad (18)$$

Решение оптимизационной задачи возможно простым численным методом – методом простых итераций (ЕМ-алгоритм).

Есть 2 очень известных частных случая решения.

1. RLSA, вероятностный латентный семантический анализ, при котором $R(\Phi, \Theta) = 0$.

2. LDA, латентное размещение Дирихле. Уникальная особенность моделей LDA состоит в том, что темы не обязательно должны быть различными и слова могут встречаться в нескольких темах; это придает некоторую нечеткость определяемым темам, что может пригодиться для совладания с гибкостью языка [12].

Подход аддитивной регуляризации ARTM (Additive Regularization for Topic Modeling) основан на идее многокритериальной регуляризации. Он позволяет строить модели, удовлетворяющие многим ограничениям одновременно. Каждое ограничение формализуется в виде регуляризатора, зависящего от параметров модели. Затем взвешенная сумма всех таких критериев максимизируется совместно с основным критерием правдоподобия.

Хорошей библиотекой для тематического моделирования с аддитивной регуляризацией является BigARTM. В ней реализовано несколько механизмов, которые снимают многие ограничения простых моделей типа PLSA или LDA и расширяют спектр приложений тематического моделирования.

2.3. Обзор нейросетевых моделей обработки естественного языка

Нейросетевые модели обычно способны выделять более сложные паттерны – взаимоотношения в данных. В случае текстов (а именно, отдельных предложений) такие паттерны проявляются в том, как именно слова употребляются вместе, как построены фразы.

В данной работе были исследованы следующие нейросетевые модели обработки естественного языка применительно к задаче многоклассовой классификации: сверточная нейронная сеть (CNN), нейронная сеть LSTM и нейронная сеть на основе механизмов внимания – трансформер BERT.

Сверточные нейронные сети

Свертка – это некоторая операция, выполняемая над двумя функциями. В контексте нейросетей, первый сигнал – это входные данные, а второй сигнал – это часть параметров нейросети, которые выполняют роль ядра свертки.

Сначала мы строим матрицу эмбедингов слов или символов. Количество строк соответствует длине текста. Затем мы поочередно перебираем смещения и, для каждого значения смещения, мы выбираем k целых строк из входной матрицы. Затем мы вытягиваем эти строки в один вектор и вычисляем скалярное произведение с ядром (рисунок 2). Ядро при этом описывает паттерн, который мы ищем.

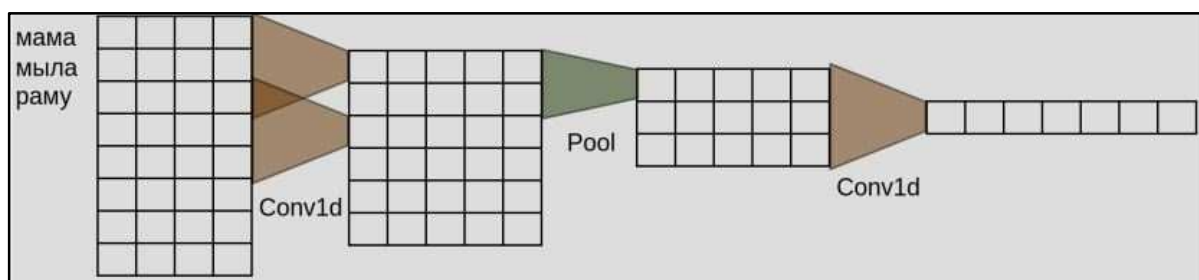


Рисунок 2 – Операции свертки и пулинга на текстовых данных

Обычно сверточные блоки применяют один за другим, перемешивая с блоками пулинга или агрегации, а также с функциями активации [13]. Делается это для того, чтобы расширить пятно восприятия, то есть получить

возможность обрабатывать более широкий контекст, более длинные паттерны.

Заставить сверточную нейросеть обрабатывать длинные предложения и учитывать широкий контекст достаточно сложно – для этого необходимо сделать очень глубокую сеть, в которой будет много параметров, что сильно усложнит процесс обучения. Чтобы расширить пятно восприятия, можно применять «разреженные свертки» (английский термин – «dilated convolutions»). Идея в том, что ядро свертки применяется не к непрерывному фрагменту сигнала, а к фрагменту, из которого удалена часть элементов (как правило, удаляют элементы с четными номерами) (рисунок 3).

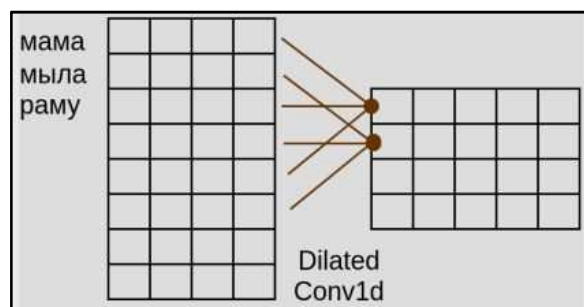


Рисунок 3 – Разреженная свертка

Таким образом почти в два раза увеличивается рецептивное поле, а количество параметров остается прежним.

Если применять разреженные свертки на первом слое, мы, фактически, будем игнорировать каждое второе слово, это совсем не то что мы хотим. Поэтому на первом слое применяют обычные, не прореженные свертки, а затем на каждом новом уровне увеличивают прореживания в два раза.

Рекуррентные нейронные сети

Другой вид нейросетей, используемых в задачах обработки текста – это рекуррентные нейросети. Основная их идея заключается в том, что у нас есть некоторое внутреннее состояние, которое мы обновляем после прочтения очередного элемента входной последовательности. Это внутреннее состояние должно содержать в себе информацию обо всем прочитанном ранее тексте. Такие нейросети потенциально гораздо мощнее, чем сверточные, но хуже распараллеливаются.

Долгая краткосрочная память (long short-term memory, LSTM) – особая разновидность архитектуры рекуррентных нейронных сетей, способная к обучению долговременным зависимостям.

Структура слоя сети LSTM показана на рисунке 4.

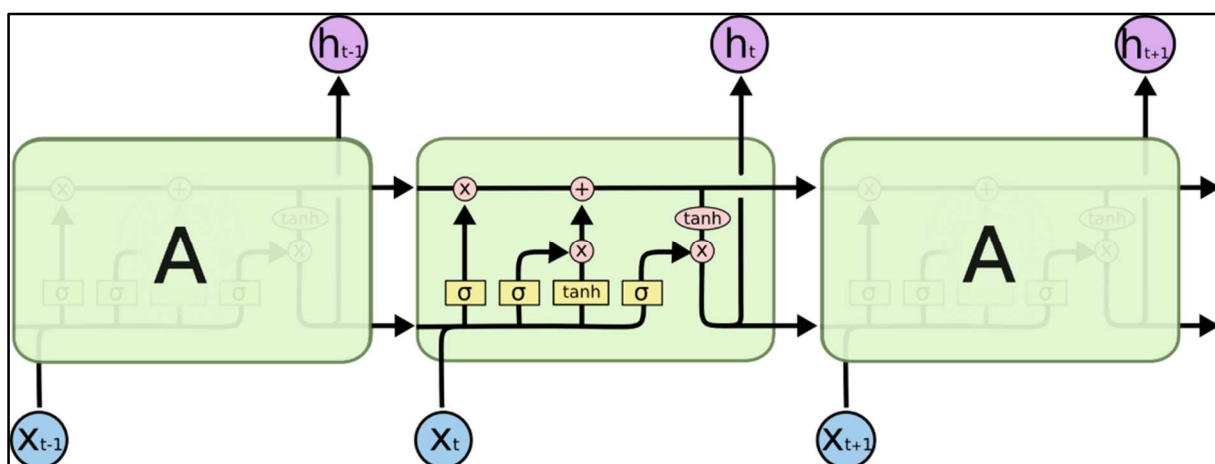


Рисунок 4 – Структура нейронной сети LSTM

Первый шаг в LSTM – определить, какую информацию можно выбросить из состояния ячейки. Это решение принимает сигмоидальный слой, называемый «слоем фильтра забывания» (forget gate layer).

Следующий шаг – решить, какая новая информация будет храниться в состоянии ячейки. Этот этап состоит из двух частей. Сначала сигмоидальный слой под названием «слой входного фильтра» (input layer gate) определяет, какие значения следует обновить. Затем \tanh -слой строит вектор новых значений-кандидатов, которые можно добавить в состояние ячейки.

Далее, к состоянию ячейки применяются некоторые фильтры. Сначала применяем сигмоидальный слой, который решает, какую информацию из состояния ячейки выводить. Затем значения состояния ячейки проходят через \tanh -слой, чтобы получить на выходе значения из диапазона от -1 до 1, и перемножаются с выходными значениями сигмоидального слоя, что позволяет выводить только требуемую информацию.

Нейронные сети на основе механизма внимания

Другая архитектура нейронной сети, протестированная мной в данной работе, носит название «трансформер» (transformer). Она была представлена в 2017 году в статье с программным названием «Внимание – это все, что вам нужно» [14], написанной группой исследователей из Google Brain и Google Research. Быстрое развитие сетей, основанных на трансформерах, привело к появлению гигантских языковых моделей, таких как Generative Pre-trained Transformer 3 (GPT-3) от OpenAI, или Bidirectional Encoder Representations from Transformers (BERT) от Google, способных эффективно решать множество задач из области NLP.

Агрегация (пулинг) обычно применяется, когда нужно получить представление информации вне зависимости от того, где конкретно во входных данных эта информация встречалась. При этом пространственная информация теряется частично или полностью (в случае глобального пулинга). Основной недостаток пулинга – в том, что каждый канал агрегируется независимо. Амплитуда значения отвечает как за саму информацию, так и за ее значимость. Гораздо более эффективно будет, если разделить операции оценки значимости информации (привлечения внимания) и ее использование. На рисунке 5 показана одна из возможных структурных схем работы механизма внимания.

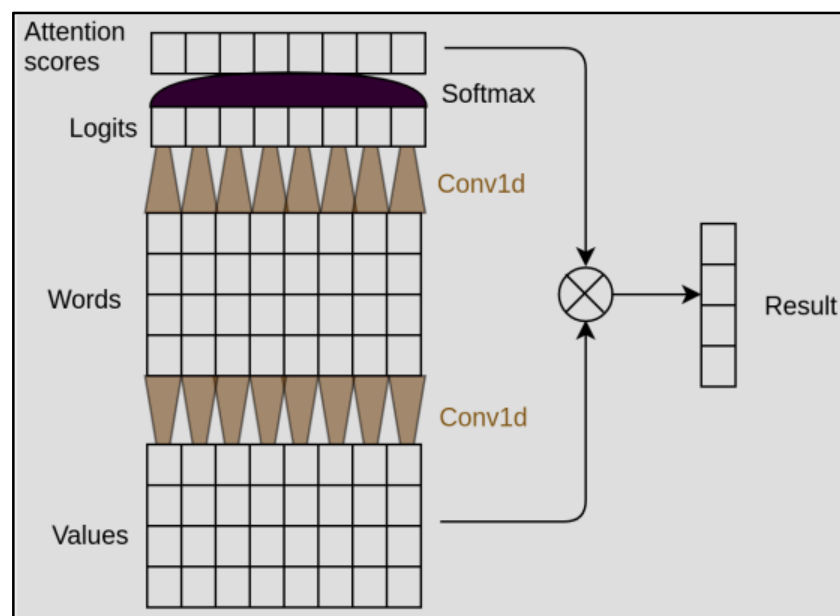


Рисунок 5 – Структурная схема работы механизма внимания

К вектору каждого слова независимо применяется однослойная нейросеть, выход которой представляет собой оценку значимости слова. Оценки релевантности слов нормируются (softmax). Затем они домножаются на соответствующие значения входных векторов, и поэлементно складываются, и мы получаем результирующий вектор. Предварительно значения входных векторов можно дополнительно преобразовать с помощью еще одной нейросети. В результате получаем намного большую гибкость, чем при обычной агрегации с помощью пулинга: с ростом длины последовательности внимание не начинает работать хуже.

Часто релевантность слов надо оценить в контексте конкретной потребности пользователя на основе механизма запроса. Но можно сформировать такой запрос и внутри механизма внимания. В этом случае, если в качестве ключей и значений используются элементы одного и того же текста, механизм называется механизмом внутреннего внимания (self-attention) (рисунок 6).

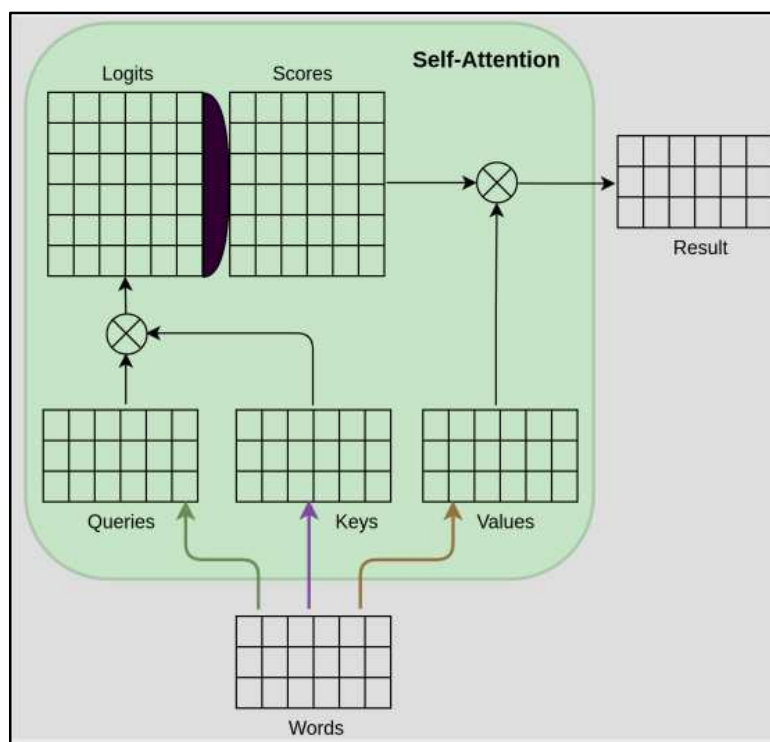


Рисунок 6 – Механизм внутреннего внимания

Для получения матриц Queries, Keys и Values часто используется простое линейное преобразование – умножение матрицы эмбеддингов на матрицы проекций.

В усовершенствованном виде механизм внимания включает несколько голов, где в каждой голове матрицы проекций имеют разные веса (Multi-Head Self-Attention). Это позволяет строить результирующий вектор признаков, учитывая сразу множество аспектов. Такой вариант внутреннего внимания используется в трансформерах.

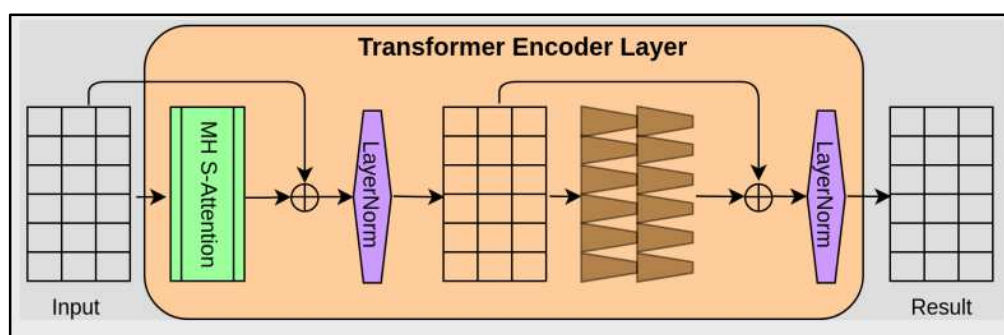


Рисунок 7 – Структура слоя трансформера

Трансформер состоит из нескольких слоев следующего вида (рисунок 7):

- 1) Multi-Head Self-Attention;
- 2) сверточные слои (нелинейные преобразования);
- 3) связи в обход нелинейностей.

Информация о порядке слов в тексте в таких моделях обычно закладывается в эмбединги.

2.4. Метрики качества моделей

Для оценки качества моделей обработки естественного языка были выбраны следующие метрики качества: *accuracy*, *precision* (точность), *recall* (полнота), *F1* и *MCC* (коэффициент корреляции Мэтьюза). Данные метрики качества позволяют как отобрать лучшие модели в ходе экспериментов, так и оценить качество разработанного решения.

Метрика *accuracy* является базовой метрикой, которая измеряет количество верно классифицированных объектов относительно общего количества всех объектов. Однако она имеет недостаток – она не подходит для несбалансированных классов. Метрика *accuracy* вычисляется по формуле (19):

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \quad (19)$$

где *TP* – доля объектов, верно отнесенных к положительному классу;

TN – доля объектов, верно отнесенных к отрицательному классу;

FP – доля объектов, неверно отнесенных к положительному классу;

FN – доля объектов, неверно отнесенных к отрицательному классу.

Метрика *precision* (точность) показывает долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, и рассчитывается по формуле (20):

$$precision = \frac{TP}{TP + FP}. \quad (20)$$

Метрика *recall* (полнота) показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм, и рассчитывается по формуле (21):

$$recall = \frac{TP}{TP + FN}. \quad (21)$$

Чем выше точность и полнота, тем лучше. Но в реальной жизни максимальная точность и полнота недостижимы одновременно и приходится искать некий баланс. Метрика *F1* является средним гармоническим между *precision* и *recall*, применяется для общей оценки качества классификатора и рассчитывается по формуле (22):

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \quad (22)$$

Метрика *MCC* (коэффициент корреляции Мэтьюза) обычно рассматривается как сбалансированная мера, которую можно использовать, даже если мы имеем классы очень разных размеров. *MCC* возвращает значение от -1 до +1. Коэффициент +1 представляет собой идеальное предсказание, 0 не лучше, чем случайное предсказание, а -1 указывает на полное несоответствие между предсказанием и наблюдением.

Метрика *MCC* вычисляется по формуле (23):

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}. \quad (23)$$

Для задач многоклассовой классификации существует несколько способов вычисления метрик *precision*, *recall* и *F1*.

1. Macro-avg: метрики рассчитываются отдельно для каждого класса, а затем находится их невзвешенное среднее. Данный способ подходит, если каждый класс имеет одинаковую важность и результат не должен быть искажен в пользу любого из классов в наборе данных. Данный способ не учитывает дисбаланс классов в конкретном наборе данных.

2. Micro-avg: показатели вычисляются глобально. Подсчитываются следующие величины:

$$TP = TP_1 + \dots + TP_k,$$

$$TN = TN_1 + \dots + TN_k,$$

$$FP = FP_1 + \dots + FP_k,$$

$$FN = FN_1 + \dots + FN_k,$$

где k – количество классов. Затем рассчитанные значения подставляются в формулы (20) – (22). Этот критерий может быть полезен, если имеется заинтересованность в общей статистике с точки зрения TP , TN , FP , FN и мы не обеспокоены наличием различий внутри классов.

3. **Weighted**: рассчитываются метрики для каждого класса и находится их средневзвешенное значение по поддержке (количеству образцов данного класса). Данный способ подходит, если важность классов пропорциональна количеству образцов, т. е. класс, который недостаточно представлен, считается менее важным.

4. **Samples**: вычисляются метрики для каждого объекта и находится их среднее значение. Имеет смысл только для многозначной классификации.

В задаче анализа обращений граждан будем опираться на средневзвешенные значения метрик (**weighted**), так как мы имеем неравномерное распределение классов в наборе данных.

Выводы по разделу 2

В данном разделе были рассмотрены основные архитектуры статистических и нейросетевых моделей для анализа текстовых данных, используемые на сегодняшний день и хорошо зарекомендовавшие себя.

В процессе реализации системы описанные модели будут обучены на исходном наборе данных и применены для решения задачи анализа обращений граждан.

3. ПРОЕКТИРОВАНИЕ

3.1. Требования к системе

На основе анализа предметной области были выявлены следующие функциональные и нефункциональные требования к системе.

Функциональные требования

1. Система должна поддерживать многопользовательский режим.
2. Система должна предоставлять интерфейс для загрузки текста обращений граждан.
3. Система должна для каждого обращения рассчитывать следующие результаты классификации и предоставлять интерфейс для их вывода пользователю:
 - категорию обращения (топ-3 вероятных класса);
 - тему обращения (топ-3 вероятных класса);
 - ключевые слова найденных тем;
 - предполагаемого исполнителя (топ-3 вероятных класса).
4. Система должна предоставлять интерфейс для просмотра обращений граждан.

Нефункциональные требования

1. Система должна иметь возможность запуска на Linux сервере.
2. Система должна быть реализована с помощью языка Python.
3. Система должна хранить обращения граждан в базе данных.

3.2. Варианты использования системы

На основе списка требований, предъявляемых к проектируемой системе, были разработаны варианты использования системы, которые должны быть доступны через пользовательский интерфейс. Диаграмма вариантов использования системы представлена на рисунке 8. С системой взаимодействуют 2 актера, описанные далее.



Рисунок 8 – Диаграмма вариантов использования системы

Пользователь – пользователь системы, который может выполнять следующие действия из графического интерфейса системы.

1. Выполнить анализ обращения. Этот вариант использования включает в себя такой вариант использования, как ввод текста обращения в форму. В графическом интерфейсе должны отобразиться результаты анализа.

2. Просмотреть историю своих запросов к системе, нажав кнопку «Отобразить» на боковой панели. Вариант использования может быть расширен вариантом использования «Просмотреть историю запросов по условию» в точке расширения «Формирование запроса». В таком случае, пользователь может выбрать количество отображаемых запросов.

3. Вывести статистику своих запросов. В данном варианте использования отображается количество запросов пользователя.

4. Скрыть историю своих запросов. В данном варианте использования все внесенные пользователем обращения помечаются системой как недоступные к просмотру данным пользователем.

5. Зарегистрировать нового пользователя. В любой момент можно создать нового пользователя, предварительно указав логин и пароль.

Администратор – администратор системы, которому доступны все действия пользователя. Кроме того, администратор системы может просматривать историю запросов всех пользователей, статистику по обращениям к системе всех пользователей, а также может очистить историю всех запросов пользователей.

3.3. Проектирование архитектуры системы

Общая архитектура системы представлена в виде диаграммы размещения (развертывания) на рисунке 9, на которой отражены ключевые компоненты и их место в общей структуре.

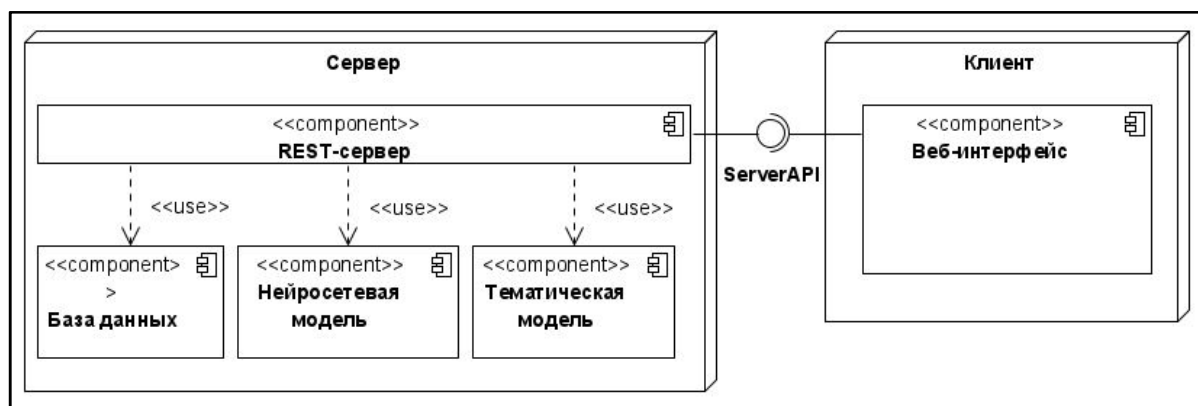


Рисунок 9 – Диаграмма размещения системы

Система состоит из двух узлов – сервер и клиент. Взаимодействие между узлами осуществляется через API сервера. Сервер включает в себя 3 компонента, а клиент – 1 компонент. Рассмотрим их подробнее.

Компонент «REST-сервер» предоставляет API для взаимодействия клиента и сервера, отвечает за запросы загрузки на сервер текста обращений

граждан, и предоставления клиенту результатов анализа. Он взаимодействует с базой данных SQLite3, нейросетевой моделью классификации BERT-pytorch и тематической моделью на основе аддитивной регуляризации BigARTM.

Компонент «База данных» осуществляет хранение всех анализируемых пользователем обращений граждан и информацию о них. Rest-сервер взаимодействует с базой данных SQLite3, запрашивая и добавляя информацию в нее.

Компонент «Нейросетевая модель» осуществляет предсказание категории (классификацию) обращения на основе модели машинного обучения BERT-pytorch.

Компонент «Тематическая модель» осуществляет предсказание темы обращения, 10 ключевых слов данной темы и топ-3 исполнителей на основе тематической модели с аддитивной регуляризацией BigARTM.

Компонент «Веб-интерфейс» реализует графический интерфейс и обеспечивает взаимодействие пользователя с системой.

3.4. Проектирование серверной части

Согласно требованиям к системе REST-сервер должен предоставлять следующий API для взаимодействия пользователя с системой:

- 1) POST /message - загрузка новых обращений граждан в систему;
- 2) GET /message – выгрузка заданного числа последних обращений из системы;
- 3) DELETE /message – помечает все обращения, загруженные данным пользователем, недоступными к просмотру данным пользователем;
- 4) GET /statistics - возвращает информацию об обращениях граждан, загруженных в систему;
- 5) GET /categories – возвращает топ-3 вероятных категорий из последнего запроса;

6) GET /topwords – возвращает 10 ключевых слов темы последнего запроса;

7) GET /executor – возвращает топ-3 исполнителей последнего запроса;

8) POST /newuser – создает нового пользователя в системе, в случае если данный логин еще не был занят.

С каждым запросом передается логин и пароль пользователя. По умолчанию, для упрощения демонстрации интерфейса, передается логин и пароль «test».

Администратору доступны для отображения сообщения всех пользователей, и только он может окончательно удалить загруженные в систему обращения граждан.

Для хранения лога результатов обработки предусмотрено использование реляционной базы данных, схема которой представлена на рисунке 10.

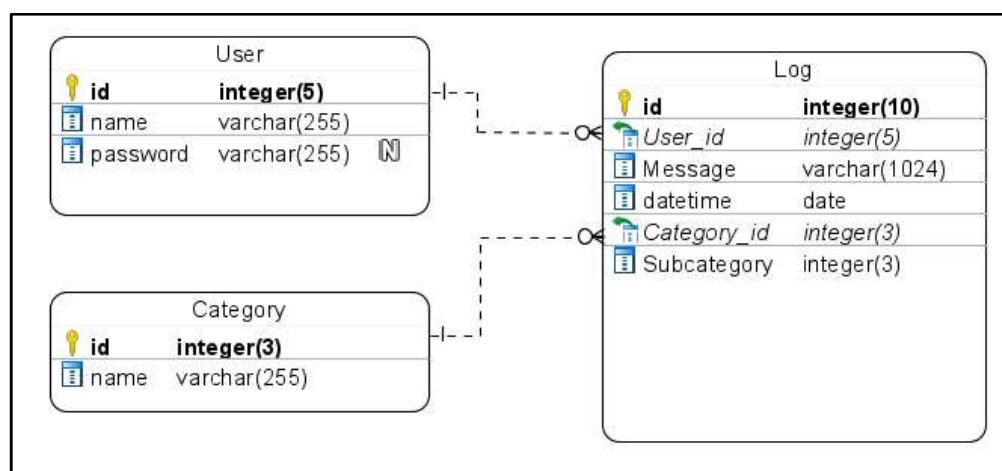


Рисунок 10 – Схема базы данных

В базе данных имеются следующие таблицы:

1) User – таблица содержит имена пользователей, пароли (опционально) и идентификаторы пользователей;

2) Category – таблица содержит перечень всех категорий и их идентификаторы;

3) Log – таблица содержит результаты каждого анализа: идентификатор пользователя, текст обращения, идентификатор присвоенной данному обращению категории, тема обращения, дата и время запроса.

3.5. Проектирование микросервиса на архитектуре REST

На рисунке 11 представлена диаграмма деятельности для обработки запросов сервером.

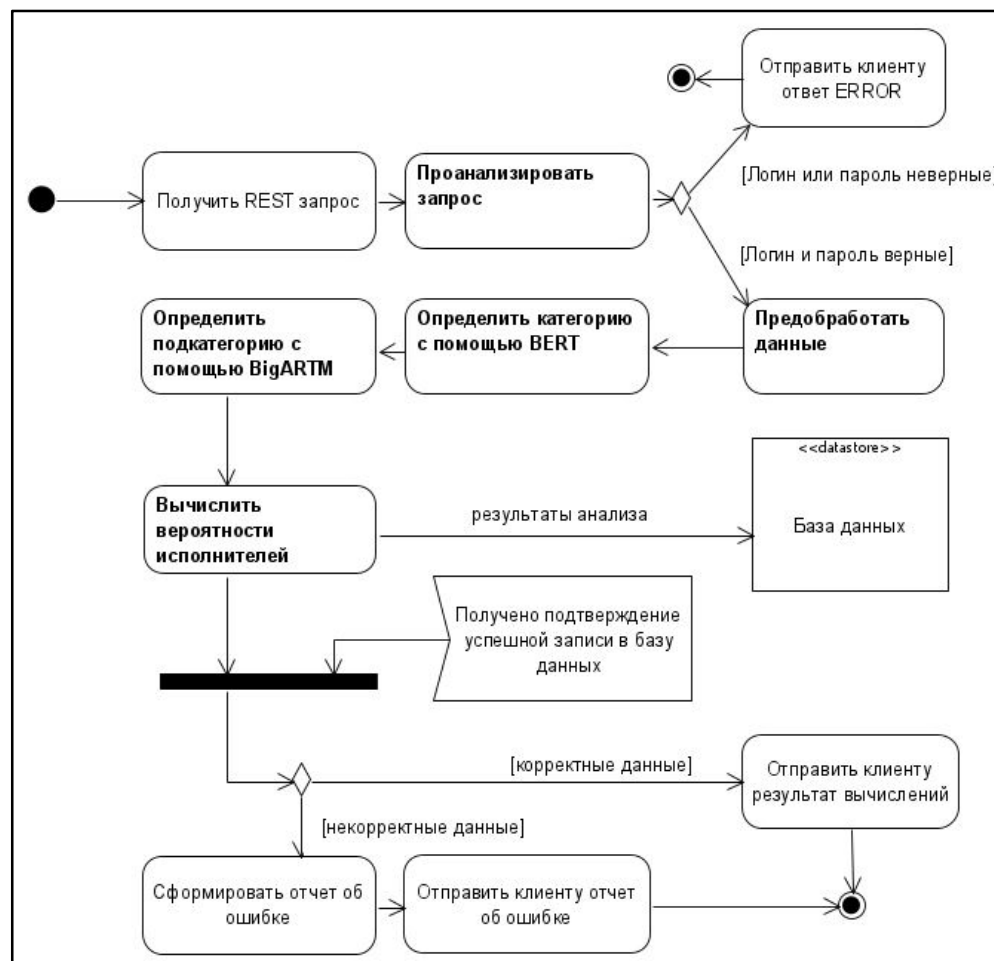


Рисунок 11 – Диаграмма деятельности обработки запросов сервером

Данная диаграмма построена для варианта использования системы «Выполнить анализ обращения». Получив REST запрос, сервер начинает его декомпозицию и анализ. Если логин или пароль неверные, сервер отправляет клиенту ответ с сообщением об ошибке. Если пароль и логин

верны, сервер начинает предобработку данных. Далее с помощью нейросетевой модели BERT определяется категория обращения, с помощью иерархической тематической модели BigARTM определяется тема обращения. По полученным данным вычисляются наиболее вероятные исполнители. Полученные результаты заносятся в базу данных.

При подтверждении успешной записи в базу данных и в случае корректного результата полученный результат анализа обращения отправляется клиенту. В противном случае, клиенту отправляется сформированный отчет об ошибке.

3.6. Проектирование интерфейса пользователя

На рисунке 12 представлена диаграмма состояний объекта интерфейса пользователя. Диаграмма состояний охватывает 3 состояния данного объекта. Это «Ожидание нажатия кнопки», «Активное состояние», и «Обработка ошибок», а также начальное состояние и конечное состояние. Диаграмма также отражает переходы между состояниями.

Если пользователем произведено нажатие кнопки, то объект пользовательского интерфейса переходит из состояния «Ожидание нажатия кнопки» в суперсостояние «Активное состояние», в котором сменяет состояния формирования запроса, предобработки данных, считывания данных, подтверждения ответа сервера и отображения информации пользователю.

Если в каком-то из этих состояний возникает исключение, объект переходит из суперсостояния «Активное состояние» в состояние «Обработка ошибки» и затем в конечную точку.

Если при нахождении объекта в состоянии «Ожидание нажатия кнопки» было закрыто окно пользовательского интерфейса, объект также переходит в конечную точку.

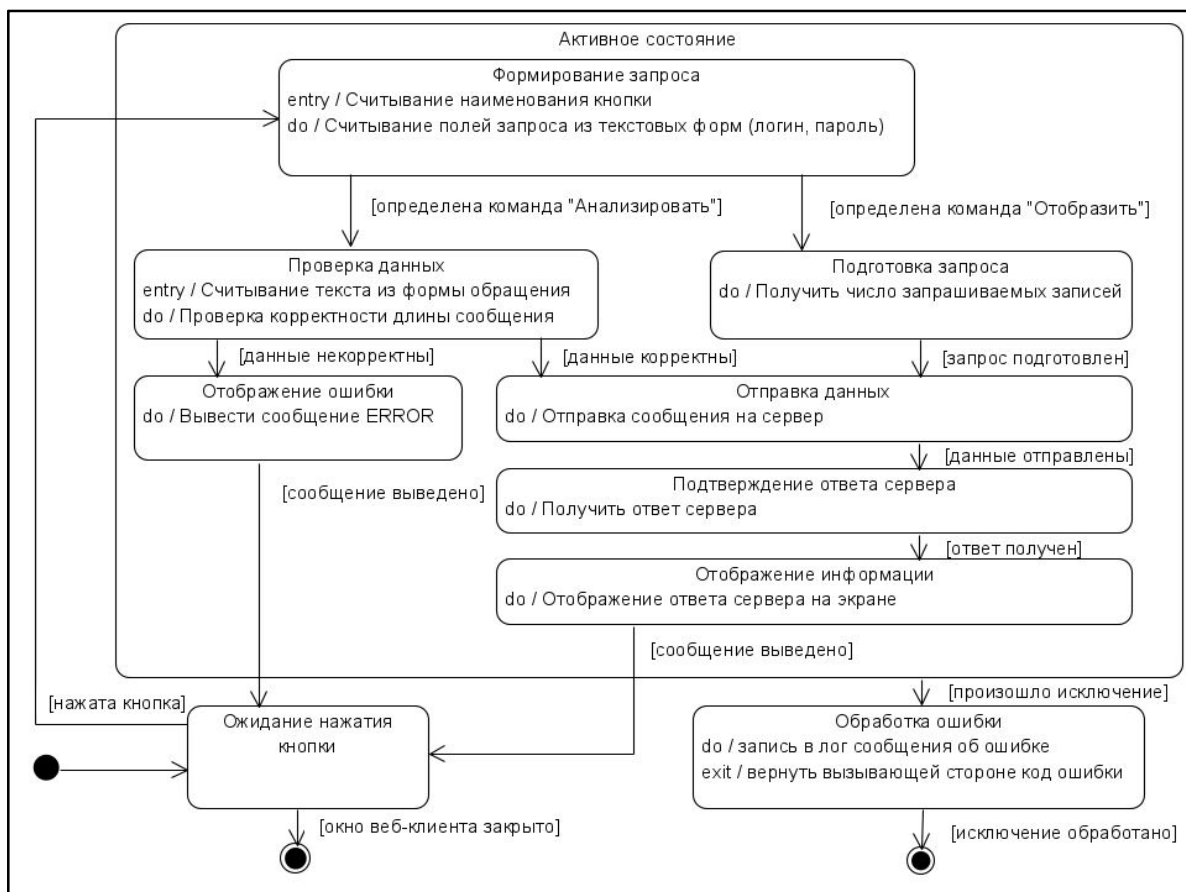


Рисунок 12 – Диаграмма состояний объекта интерфейса пользователя

Выводы по разделу 3

В данном разделе было выполнено проектирование архитектуры системы, в том числе ее серверной и клиентской части. Выделены варианты использования системы. Также построены диаграмма деятельности REST сервиса (для варианта использования системы «Выполнить анализ обращения») и диаграмма состояний объекта пользовательского интерфейса.

Полученные при проектировании системы результаты будут использованы мной при реализации системы автоматизированной обработки обращений граждан.

4. РЕАЛИЗАЦИЯ

4.1. Средства реализации

В качестве основного метода классификации текстов исследовались нейросетевые модели на основе RuBERT (реализация DeepPavlov) [15].

Использовались техники fine-tuning модели RuBERT под задачу классификации обращений на наборе данных Республики Татарстан [16].

Для извлечения именованных сущностей из текстов обращений (названий организаций, локаций, персон, дат и т.д.) был выбран вариант Slovnnet NER из библиотеки Natasha.

В качестве метода разведочного поиска для выделения дополнительных подкатегорий (тем) лучше всего себя показали иерархические тематические модели на основе аддитивной регуляризации, описанные в [17], [18].

Для обучения моделей использовались облачные ресурсы платформы Paperspace Gradient. Это одно из первых предложений ML PaaS (Platform as a Service) в отрасли, которое предлагает сквозное управление жизненным циклом моделей машинного обучения. Платформа включает в себя множество инструментов, позволяющих пройти все этапы по сопровождению модели: от разработки, обучения и настройки до развертывания.

Характеристики инстанса, используемого для обучения моделей:

- 1) RAM 30 Гб;
- 2) HDD 100 Гб;
- 3) CPU Intel(R) Xeon(R) CPU E5-2623 v4 @ 2.60GHz;
- 4) GPU NVIDIA Quadro P5000 16 Gb GDDR5X;
- 5) операционная система Ubuntu 18.04;
- 6) интерпретатор Python версии 3.6.9.

4.2. Подготовка данных для обучения

Поставленная передо мной задача является задачей классификации.

В качестве исходных данных мне был дан размеченный набор данных региона Татарстан. Набор данных представляет собой текстовый файл, содержащий тексты обращений граждан и метки, присвоенные ассессорами вручную для каждого обращения.

Данный набор данных содержит 31312 записей - текстов обращений граждан, которым присвоены 37 тематических меток категорий и 121 метка об исполнителях (82 метки осталось после предобработки).

Предобработка данных состояла из нескольких этапов.

Первым этапом выполнена предварительная нормализация, алгоритм представлен на рисунке 13.

```
text = emo_to_text(text) #преобразует смайлики в слова
text = emoji.demojize(text, delimiters=(" ", "")) #преобразует эмодзи-символы в слова
text = text.lower().replace("ё", "е")
text = re.sub('((www\.[^\s]+)|(https?:\/\/[^\s]+))', 'URL', text)
text = re.sub('@[^\s]+', 'USER', text)
text = re.sub('[^a-zA-Za-яA-Я]+', ' ', text) #удаляем цифры и спецсимволы
text = replace_three_or_more(text) #удаляет дублирование букв, если число букв более 3
text = re.sub(' +', ' ', text) #заменяем несколько пробелов одним
text = text.strip()
```

Рисунок 13 – Предварительная нормализация данных

Далее была выполнена очистка данных. Были удалены обращения, текст которых содержал менее 4-х слов. После этого набор данных содержал уже 23698 строк. Были сформированы перечни всех категорий обращений, и если в определенной категории число обращений было менее 10, то такие категории были исключены. В результате число строк в наборе данных сократилось до 23670. Алгоритм очистки данных представлен на рисунке 14.

```

data['num_words'] = data['description_prep'].apply(str.split,
args = (' '))
data['num_words'] = data['num_words'].apply(len)
data = data.loc[data['num_words'] > 3]
data.reset_index(inplace=True)
cat_num = dict()
for cat in categories:
    num = np.sum(data['categories'] == cat)
    cat_num.update({cat : num})
for key, value in cat_num.items():
    if value < 10:
        data = data.loc[data['categories'] != key]
data.reset_index(inplace=True)

```

Рисунок 14 – Очистка данных

Для удаления из текста именованных сущностей, таких как имен людей и местоположений, был использован фреймворк Natasha. Именованные сущности в тексте были заменены на соответствующие тэги – PER и LOC.

Подобным образом (рисунки 13, 14) были обработаны данные об исполнителях. Отличие состояло в том, что именованные сущности не заменялись на тэги, а окончательно удалялись. К примеру, исполнитель «Администрация г. Казани» был преобразован к виду «Администрация».

Если в последующем для создания векторной модели использовать токенизатор подслов, такой как WordPiece в BERT, можно считать текст обращений уже нормализованным. Но для статистических моделей используется упрощенное представление текста в виде «мешка слов», и чтобы сократить словарь таких моделей, желательно провести еще один этап нормализации.

Поэтому для статистических моделей данные были преобразованы в отдельный объект `pandas.Series`, для которого выполнена лемматизация. Алгоритм лемматизации приведен на рисунке 15.

```

morph = pymorphy2.MorphAnalyzer() #по умолчанию русский язык
stemmer = SnowballStemmer("english")
stops = set(stopwords.words("english")) | set(stopwords.words("russian"))
ttext_stem = list()
for row in data['description_prep']:
    words = row.split(' ')
    nf_words = list()
    for word in words:
        try:
            ru_word = re.sub('[^а-яА-Я]+', ' ', word)
            en_word = re.sub('[^a-zA-Z]+', ' ', word)
            if ru_word == word and word != '': # кириллица
                parse_word = morph.parse(word)[0]
                nf_word = parse_word.normal_form
                nf_words.append(nf_word.strip())
            elif en_word == word and word != '': # латиница
                nf_word = stemmer.stem(word)
                nf_words.append(nf_word.strip())
            else:
                pass
        except Exception as e:
            print(f'Слово не преобразовано {e}')
            continue
    nf_words = [w for w in nf_words if not w in stops]
    nf_words = ' '.join(nf_words)
    ttext_stem.append(nf_words)

```

Рисунок 15 – Лемматизация данных

Все данные сохраняются в файл csv (comma-separated values) для дальнейшего использования.

4.3. Реализация алгоритмов анализа текстовых данных

Будет рассмотрено построение статистических и нейросетевых моделей для задачи классификации обращений по категориям (задача обучения с учителем) и проанализированы полученные результаты.

Затем каждая категория будет разбита на темы. Это требуется, чтобы иметь возможность более точно определять исполнителя. Так как у нас нет меток, отвечающим темам, для каждой категории будет решена задача обучения без учителя с использованием иерархических тематических моделей на основе аддитивной регуляризации.

Логистическая регрессия

Для создания разреженной матрицы векторов использовалось невзвешенное `CountVectorizer()` и взвешенное `TfidfVectorizer()` преобразование из библиотеки `sklearn`, с тем чтобы сравнить полученные результаты. Использовался набор данных с лемматизацией. `CountVectorizer()` по умолчанию выполняет токенизацию по словам и исключает токены с длиной, меньшей чем 2 символа. Кроме того, был задан параметр `min_df = 5`, чтобы игнорировать слова, которые встречаются менее, чем в 5 документах.

Для проверки качества обученной модели была использована кросс-валидация (K-fold cross-validation, скользящий или перекрестный контроль).

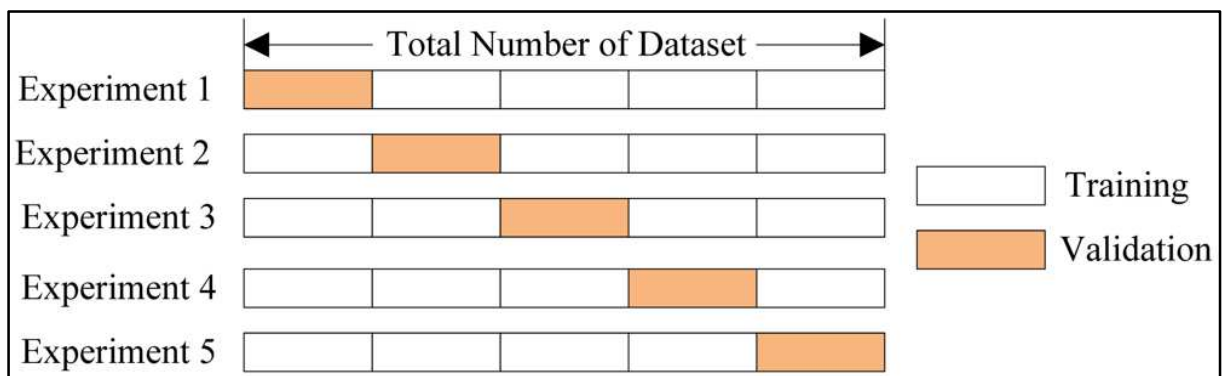


Рисунок 16 – Кросс-валидация

Все данные мы делим на 5 равных частей (рисунок 16).

Модель была обучена 5 раз на разных подвыборках исходной выборки (белый цвет), а тестировалась на одной подвыборке (каждый раз на разной, оранжевый цвет). В результате были получены 5 оценок качества модели, которые усредняются, выдавая среднюю оценку качества на кросс-валидации. Кросс-валидация обычно дает лучшую по сравнению с отложенной выборкой оценку качества модели на новых данных.

Для определения наилучших параметров регуляризации, при которых модель не переобучается, был использован поиск по параметрам `GridSearchCV()` из библиотеки `sklearn`, получено наилучшее значение $C = 1$. Получившийся код приведен на рисунке 17.

```

text_train = data['description_stem'].to_list()
vect = CountVectorizer(min_df=5).fit(text_train)
X_train = vect.transform(text_train)
y_train = data['categories'].to_list()
scoring = {'Accuracy': make_scorer(accuracy_score),
           'Precision': make_scorer(precision_score, average =
           'weighted'),
           'Recall': make_scorer(recall_score, average = 'weighted'),
           'f1': make_scorer(f1_score, average = 'weighted'),
           'mcc': make_scorer(matthews_corrcoef)}
param_grid = {'C': [0.001, 0.01, 0.1, 1, 10]}
grid = GridSearchCV(LogisticRegression(max_iter=10000), param_grid, cv=5)
grid.fit(X_train, y_train)
scores = cross_validate(LogisticRegression(max_iter=10000, C =
grid.best_params_['C']), X_train, y_train, scoring = scoring, cv=5)

```

Рисунок 17 – Обучение модели логистической регрессии

В результате получил значения метрик качества, представленные в таблице 2.

Таблица 2 – Метрики качества модели логистической регрессии

Метрика	CountVectorizer	TfIdfVectorizer
Accuracy	0.810	0.810
Precision (weighted)	0.812	0.811
Recall (weighted)	0.810	0.810
F1 (weighted)	0.805	0.803
MCC	0.729	0.727

Деревья решений

При подготовке векторов для этой и последующих статистических моделей использовалось преобразование TF-IDF. Для разбиения в вершинах дерева использовалось 2 критерия информативности – энтропийный критерий и критерий Джини. В ходе экспериментов найдена оптимальная глубина дерева – 50. Получившийся код приведен на рисунке 18.

```

for criterion in ['gini', 'entropy']:
    scores = cross_validate(tree.DecisionTreeClassifier(random_state=0, \
    criterion = criterion, max_depth = 50), X_train, y_train, \
    scoring = scoring, cv=5)

```

Рисунок 18 – Обучение модели деревьев решений

Получившиеся значения метрик качества приведены в таблице 3.

Таблица 3 – Метрики качества модели деревьев решений

Метрика	criterion = gini	criterion = entropy
Accuracy	0.728	0.707
Precision (weighted)	0.728	0.711
Recall (weighted)	0.728	0.707
F1 (weighted)	0.716	0.696
MCC	0.603	0.570

Ансамбли деревьев решений

Были применены две ансамблевых модели:

- 1) случайный лес деревьев решений;
- 2) градиентный бустинг деревьев решений.

При построении случайного леса экспериментальным способом было выбрано количество деревьев – 100, глубина каждого дерева – 50. Для разбиения в вершинах дерева использовалось 2 критерия информативности – энтропийный критерий и критерий Джини.

При построении градиентного бустинга деревьев решений экспериментально выбрана скорость обучения $learning_rate = 0.1$, количество деревьев и глубина каждого дерева такие же, как для случайного леса.

Код приведен на рисунке 19.

```
for criterion in ['gini', 'entropy']:
    scores = cross_validate(ensemble.RandomForestClassifier( \
        random_state=0, criterion = criterion, max_depth = 50,
        n_estimators = 100), X_train, y_train, scoring = scoring, cv=5)
    scores = cross_validate(ensemble.GradientBoostingClassifier( \
        random_state=0, learning_rate = 0.1, criterion = 'mse', max_depth = 50,
        n_estimators = 100), X_train, y_train, scoring = scoring, cv=5)
```

Рисунок 19 – Обучение моделей ансамблей деревьев решений

Получившиеся значения метрик качества приведены в таблице 4.

Таблица 4 – Метрики качества моделей ансамблей деревьев решений

Метрика	RandomForest, gini	RandomForest, entropy	GradientBoosting
Accuracy	0.759	0.754	0.778
Precision (weighted)	0.759	0.756	0.780
Recall (weighted)	0.759	0.755	0.778
F1 (weighted)	0.738	0.735	0.775
MCC	0.647	0.639	0.684

Метод опорных векторов (SVM)

Был использован метод опорных векторов с гауссовым ядром (RBF). Для определения наилучших параметров регуляризации, использовался поиск по параметрам GridSearchCV() из библиотеки sklearn, наилучшее значение $C = 20$.

Код приведен на рисунке 20.

```
param_grid = {'C': [0.1, 1, 5, 10, 20]}
grid = GridSearchCV(SVC(probability=False, kernel='rbf'), param_grid, \
cv=5)
grid.fit(X_train, y_train)
scores = cross_validate(SVC(probability=False, kernel='rbf', \
C = grid.best_params_['C']), X_train, y_train, scoring = scoring, cv=3)
```

Рисунок 20 – Обучение модели SVM

Получившиеся значения метрик качества приведены в таблице 5.

Таблица 5 – Метрики качества моделей ансамблей деревьев решений

Метрика	Значение
Accuracy	0.740
Precision (weighted)	0.732
Recall (weighted)	0.740
F1 (weighted)	0.716
MCC	0.617

Сверточная нейронная сеть

Для решения задачи была использована сверточная нейронная сеть с предобученным слоем Embedding. Для создания слоя Embedding была использована модель Word2Vec, обученная на русскоязычном корпусе коротких текстов RuTweetCorp [19]. Этот слой уже содержит много информации о структуре языка. В слой Embedding модели было загружено 100000 слов из словаря Word2Vec. Сначала модель была обучена с замороженным слоем Embedding, а затем данный слой был разморожен и выполнено несколько эпох финальной настройки весов.

Архитектура сверточной нейронной сети реализована на фреймворке Keras и представлена на рисунке 21.

```

input = Input(shape=(SENTENCE_LENGTH,), dtype='int32')
encoder = Embedding(NUM, DIM, input_length=SENTENCE_LENGTH, \
weights=[e_matrix], trainable=False)(input)

branches = []

x = Dropout(0.1)(encoder)
for size in [2, 3, 4, 5]:
    branch = Conv1D(filters=50, kernel_size=size, padding='valid', \
activation='relu')(x)
    branch = GlobalMaxPooling1D()(branch)
    branches.append(branch)
x = concatenate(branches, axis=1)
x = Dropout(0.1)(x)
x = Dense(300, activation='relu')(x)
output = Dense(26, activation='softmax')(x)

model = Model(inputs=[input], outputs=[output])
model.compile(loss='binary_crossentropy', optimizer='adam', \
metrics=['accuracy', precision, recall, f1])
model.summary()
checkpoint = ModelCheckpoint("cnn_weights/CNN_Word2Vec_100000-
{epoch:02d}-{val_f1:.2f}.hdf5", monitor='val_f1', save_best_only=True, \
mode='max', period=1)
history = model.fit(X_train, Y_train, batch_size=32, epochs=100, \
validation_split=0.1, callbacks = [checkpoint])

```

Рисунок 21 – Модель сверточной нейронной сети

Как видно из рисунка 21, входные данные были предварительно обрезаны до 50 слов. Далее следуют одномерные сверточные слои с размером ядра 2, 3, 4, 5, количеством фильтров 50 и операцией глобального макс-пулинга после каждого слоя. Слои прорежены (dropout = 0.1), чтобы не допустить переобучения. Далее следуют 2 полносвязных нейронных слоя. В модели использовалась функция активации relu, а в последнем слое – функция softmax. Данная архитектура неплохо зарекомендовала себя на задачах обработки текстовых данных.

Набор данных был разбит на обучающую и тестовую выборку в отношении 80:20, на валидационную выборку было взято 10% от обучающей.

На рисунке 22 показан процесс обучения данной нейросети. Функция потерь – дискретная кросс-энтропия. Данная функция потерь обычно используется в задаче многоклассовой классификации, при которой выходные значения интерпретируются как предсказания вероятностей принадлежности определенным классам [20].

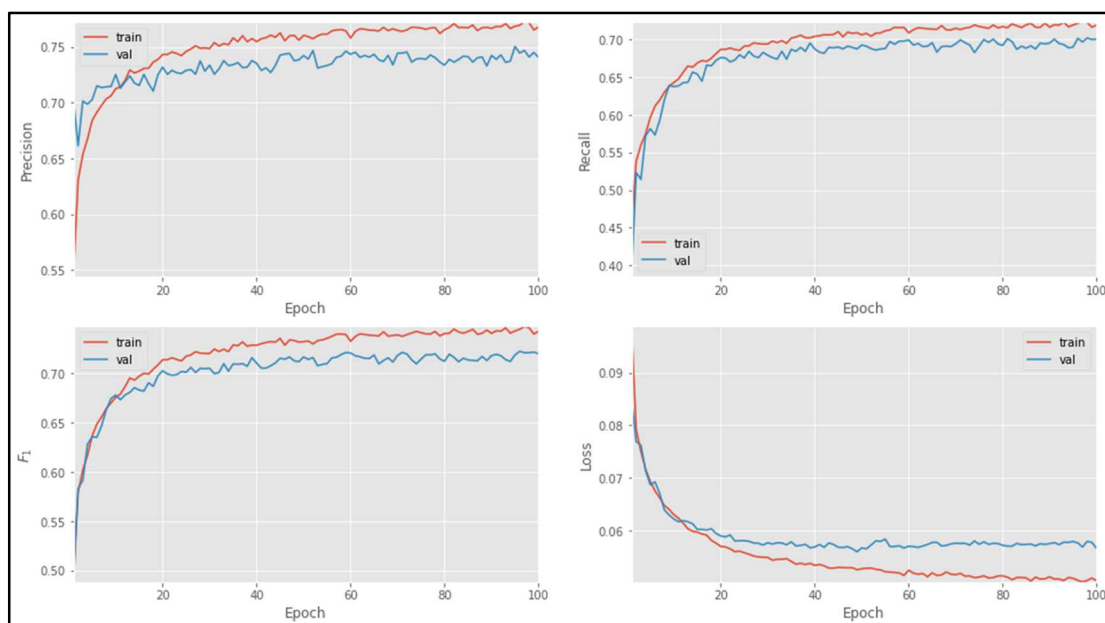


Рисунок 22 – Обучение сверточной нейронной сети

Берем лучшие сохраненные веса модели после 96 эпохи.

Далее размораживаем слой Embedding и дообучаем модель еще на 30 эпохах с параметром `learning_rate = 0.001`. Результат обучения показан на рисунке 23.

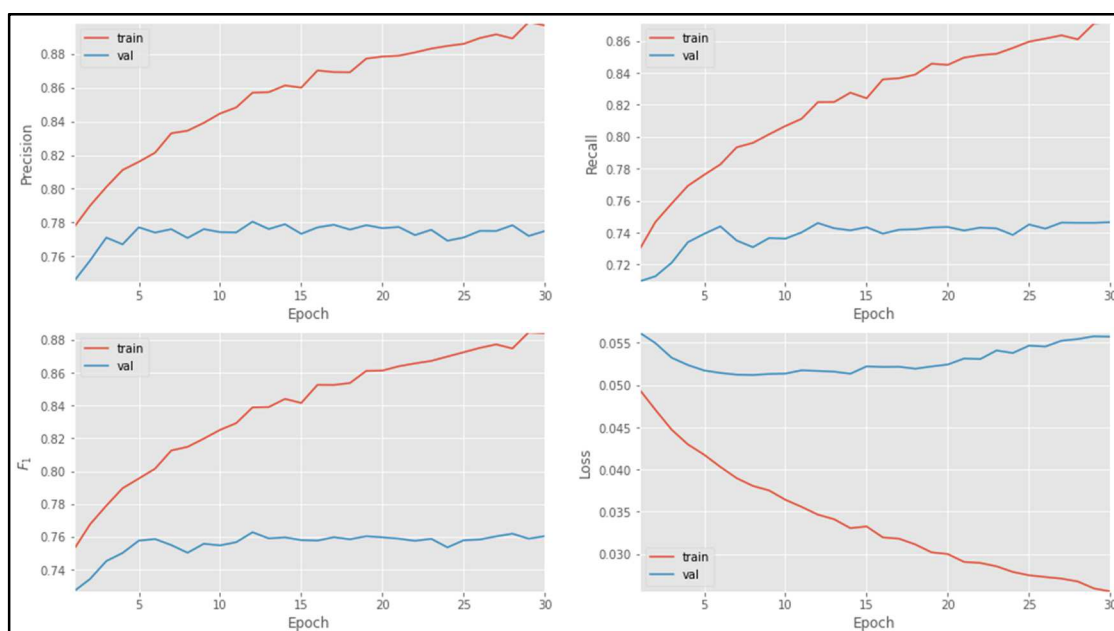


Рисунок 23 – Дообучение сверточной нейронной сети

Как видно из рисунка 23, функция потерь на валидационной выборке перестает уменьшаться с 7 эпохи, а на 12 эпохе зафиксирован максимум по метрике F1, поэтому были выбраны веса модели по результатам этой эпохи обучения.

Также была обучена эта же архитектура сверточной сети, используя токенизатор подслов WordPiece из русскоязычной модели трансформера BERT. Был загружен словарь BERT в слой Embedding и повторены действия, описанные ранее. На этот раз слой Embedding не был заморожен.

В таблице 6 приведены полученные метрики качества полученных моделей.

Таблица 6 – Метрики качества сверточной нейронной сети

Метрика	Word2Vec + CNN	WordPiece + CNN
Accuracy	0.772	0.819
Precision (weighted)	0.768	0.809
Recall (weighted)	0.772	0.819
F1 (weighted)	0.767	0.812
MCC	0.671	0.738

Рекуррентная нейронная сеть LSTM

Так как сверточная нейронная сеть с токенизатором подслов зарекомендовала себя намного лучше (таблица 6), то в ходе дальнейших исследований был использован данный токенизатор. Модель нейронной сети LSTM приведена на рисунке 24.

```

SENTENCE_LENGTH = 50
NUM = len(tokenizer.vocab)
DIM = 256
_input = Input(shape=(SENTENCE_LENGTH,), dtype='int32')
_encoder = Embedding(NUM, DIM, input_length=SENTENCE_LENGTH)(_input)
layer = LSTM(300)(_encoder)
layer = Dense(300, name='FC1')(layer)
layer = Activation('relu')(layer)
layer = Dropout(0.5)(layer)
layer = Dense(26, name='out_layer')(layer)
layer = Activation('softmax')(layer)
model = Model(_input, outputs=layer)
model.compile(loss='binary_crossentropy', optimizer='adam', \
metrics=['accuracy', precision, recall, f1])
model.summary()

```

Рисунок 24 – Модель нейронной сети LSTM

Результат обучения представлен на рисунке 25.

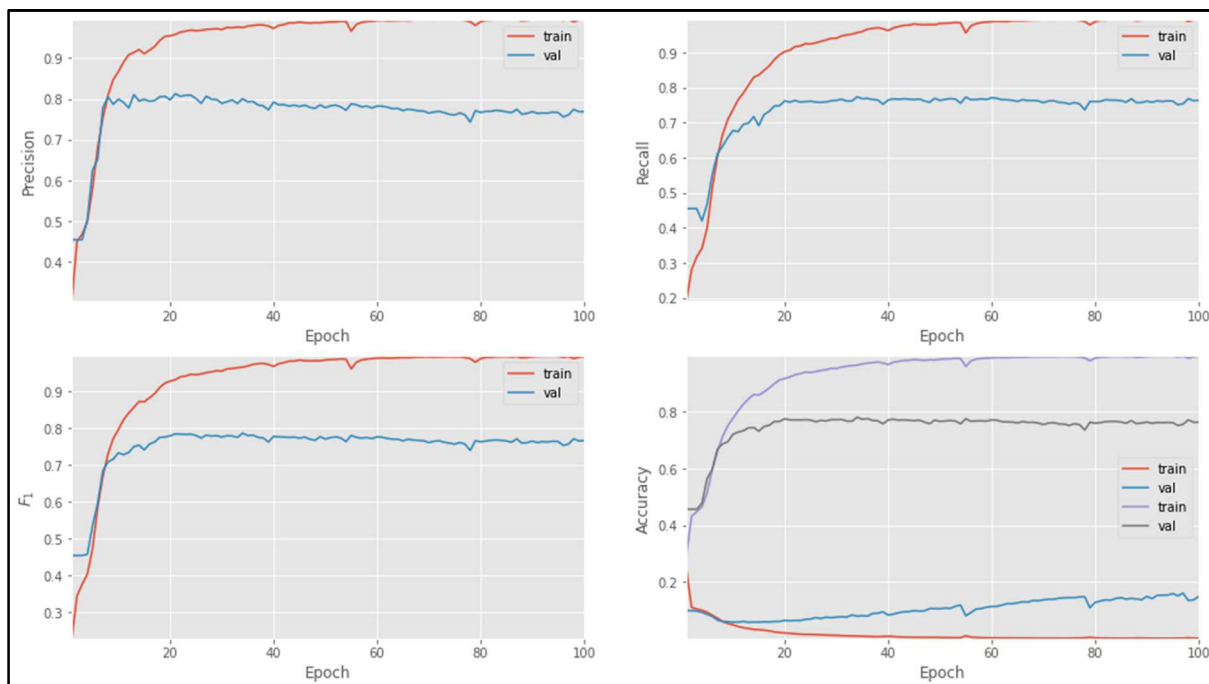


Рисунок 25 – Обучение нейронной сети LSTM

Были использованы веса с 34 эпохи обучения. В таблице 7 приведены полученные метрики качества модели.

Таблица 7 – Метрики качества нейронной сети LSTM

Метрика	Значение
Accuracy	0.764
Precision (weighted)	0.758
Recall (weighted)	0.764
F1 (weighted)	0.760
MCC	0.667

Трансформер BERT

Нейронные сети на основе механизма внимания произвели революцию в сфере NLP, сильно улучшив качество нейросетевых моделей. В открытом доступе [21], [22] имеется большое количество готовых обученных моделей, для которых нужно лишь провести тонкую настройку под собственную задачу.

Преимущества подхода на основе предобученной модели трансформера по сравнению с обучением модели «с нуля» следующие:

- 1) сокращение времени на решение задачи;
- 2) требуется меньше данных для обучения;
- 3) лучшие результаты.

Фактически, авторы рекомендуют только 2-4 эпохи обучения для точной настройки BERT для конкретной задачи NLP.

Для настройки BERT был использован фреймворк PyTorch.

BERT требует предобработки входных данных:

- 1) нужно пометить начало и конец каждого предложения маркерами [CLS] и [SEP] соответственно;
- 2) токенизировать текст с помощью WordPiece;
- 3) предоставить BERT специальный формат входных данных:
 - input ids: последовательность чисел, отождествляющих каждый токен с его номером в словаре;
 - labels: выходные метки;
 - segment mask: последовательность нулей и единиц, которая показывает, состоит ли входной текст из одного или двух предложений. Для случая одного предложения получится вектор из одних нулей;
 - attention mask: последовательность нулей и единиц, где единицы обозначают токены предложения, а нули - паддинг.

Код для одной эпохи обучения модели приведен на рисунке 26.

```
total_train_loss = 0
model.train()
for step, batch in enumerate(train_dataloader):
    # добавляем батч для вычисления на GPU
    batch = tuple(t.to(device) for t in batch)
    b_input_ids, b_input_mask, b_labels = batch
    # если не сделать .zero_grad(), градиенты будут накапливаться
    optimizer.zero_grad()
    # Forward pass
    (loss, logits) = model(b_input_ids, token_type_ids=None, \
attention_mask=b_input_mask, labels=b_labels)
    train_loss_set.append(loss.item())
    total_train_loss += loss.item()
    # Backward pass
    loss.backward()
    # Обновляем параметры и делаем шаг используя посчитанные градиенты
    optimizer.step()
    scheduler.step()
avg_train_loss = total_train_loss / len(train_dataloader)
```

Рисунок 26 – Дообучение модели BERT (одна эпоха)

Код для валидации модели приведен на рисунке 27.

```
model.eval()
# Tracking variables
total_eval_accuracy = 0
total_eval_loss = 0
nb_eval_steps = 0
valid_preds, valid_labels = [], []

for batch in validation_dataloader:
    # добавляем батч для вычисления на GPU
    batch = tuple(t.to(device) for t in batch)
    # Распаковываем данные из dataloader
    b_input_ids, b_input_mask, b_labels = batch
    # При использовании .no_grad() модель не будет считать и хранить гра-
    # диенты. Это ускорит процесс предсказания меток для валидационных данных.
    with torch.no_grad():
        (loss, logits) = model(b_input_ids, token_type_ids=None, \
attention_mask=b_input_mask, labels=b_labels)
        # Accumulate the validation loss.
        total_eval_loss += loss.item()
        # Перемещаем logits и метки классов на CPU для дальнейшей работы
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()
        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)
# Report the final accuracy for this validation run.
avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
# Calculate the average loss over all of the batches.
avg_val_loss = total_eval_loss / len(validation_dataloader)
```

Рисунок 26 – Валидация модели BERT после каждой эпохи обучения

При обучении был использован оптимизационный алгоритм стахасти-ческого градиентного спуска AdamW [23]. Скорость обучения подобрана экспериментально: $\text{learning_rate} = 2e-5$, количество эпох – 4.

Функция потерь при валидации – более надежная мера, чем точность, потому что для точности нам не важно точное выходное значение, а только то, в какую категорию оно попадает. Если мы предсказываем правильный ответ, но с меньшей уверенностью, то функция потерь покажет это, а точ-ность - нет.

На рисунке 27 приведен график функции потерь во время обучения модели. Я ориентировался на этот график при подборе количества эпох и скорости обучения. Как видно из графика, если обучать больше 4-х эпох, то функция потерь на валидационной выборке будет только расти.

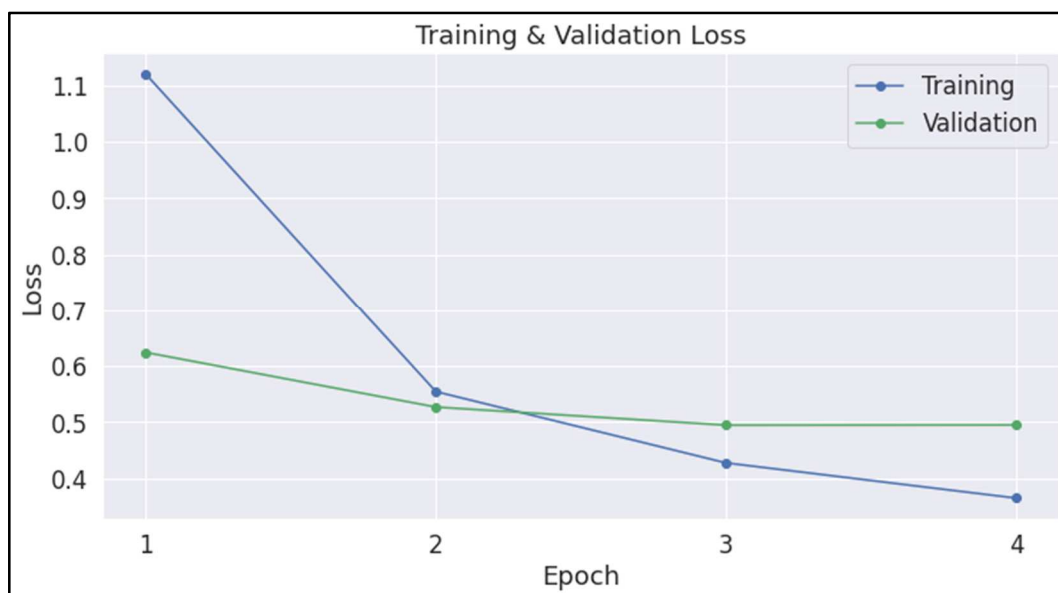


Рисунок 27 – График функции потерь при обучении модели

Затем качество модели было проверено на отложенной выборке. Результаты для различных моделей BERT приведены в таблице 8.

Таблица 8 – Метрики качества моделей на основе BERT

Метрика	sentence_ru_cased	sentence_multi_cased	sentence_base_uncased
Accuracy	0.853	0.843	0.735
Precision (weighted)	0.846	0.835	0.721
Recall (weighted)	0.853	0.843	0.735
F1 (weighted)	0.845	0.838	0.723
MCC	0.789	0.775	0.622

Тематическая модель BigARTM

Для выделения тем в каждой категории обращений была применена библиотека BigARTM.

Для начала создается словарь модели на основе набора данных. Инициализируется объект модели указанием числа подтем и списка модельностей (объектов в тексте, которым нужно присвоить разные веса). В моем случае есть только одна модальность «text». В модель вводятся метрики, по которым можно понимать, насколько хорошо она обучилась.

На рисунке 28 приведен код инициализации и обучения модели.


```

T = 10 #количество тем
model_artm = artm.ARTM(num_topics = T, topic_names = \
['sbj' + str(i) for i in range(T)], class_ids = {'text':1})
dictionary = artm.Dictionary()
dictionary.gather(data_path=data_path)
model_artm.scores.add(artm.PerplexityScore(name = 'PerplexityScore', \
dictionary = dictionary))
model_artm.scores.add(artm.SparsityPhiScore(name = 'SparsityPhiScore', \
class_id = 'text'))
model_artm.scores.add(artm.SparsityThetaScore(name = \
'SparsityThetaScore'))
model_artm.scores.add(artm.TopTokensScore(name = 'top_words', \
num_tokens = 10, class_id = 'text'))
model_artm.initialize(dictionary) #создание матриц Phi и theta
model_artm.fit_offline(batch_vectorizer = batch_vectorizer, \
num_collection_passes = 50)

```

Рисунок 28 – Инициализация и обучение модели BigARTM

В модель были введены следующие метрики.

1. PerplexityScore – метрика, позволяющая понять, насколько хорошо модель описывает данные. Чем меньше ее значение, тем лучше.

2. SparsityPhiScore – метрика разреженности матрицы Φ и SparsityThetaScore – метрика разреженности матрицы Θ . Если слово входит в небольшое количество тем, то матрица Φ разреженная. Если документ относится только к небольшому количеству тем, то матрица Θ разреженная. Мы должны добиваться как можно больших значений разреженности этих матриц, для чего в дальнейшем вводится регуляризация.

3. TopTokensScore – топы слов темы для ее интерпретации.

Для обучения модели в BigARTM есть два метода. Это fit_offline и fit_online. На маленьких коллекциях (а у нас маленькая коллекция) удобно использовать offline-обучение. В результате offline-обучения BigARTM проходит по всей коллекции много раз. Обычно рекомендуется делать 30–40 проходов по коллекции, чтобы модель сошлась. Построив график перплексии, можно оценить, насколько хорошо сошлась модель.

В темах много общеупотребительных слов (т.н. фоновой лексики). Необходимо использовать разреживающий регуляризатор матрицы Φ .

Код представлен на рисунке 29.

```

model_artm.regularizers.add(artm.SmoothSparsePhiRegularizer( \
name = 'SparsePhi', tau = -100, dictionary = dictionary, \
class_ids = ['text']), overwrite = True)
model_artm.regularizers['SparsePhi'].tau = -5*1e4
model_artm.fit_offline(batch_vectorizer = batch_vectorizer, \
num_collection_passes = 20)
top_tokens = model_artm.score_tracker['top_words']

```

Рисунок 29 – Введение разреживающего регуляризатора

После регуляризации матрицы Φ и Θ становятся более разреженными и темы лучше разделяются. Для каждой темы был выведен топ-10 слов.

Таким образом, была построена BigARTM модель для каждой категории обращений из набора данных, сохраняю эти модели и топы слов.

Алгоритм построения модели BigARTM приведен на рисунке 30.

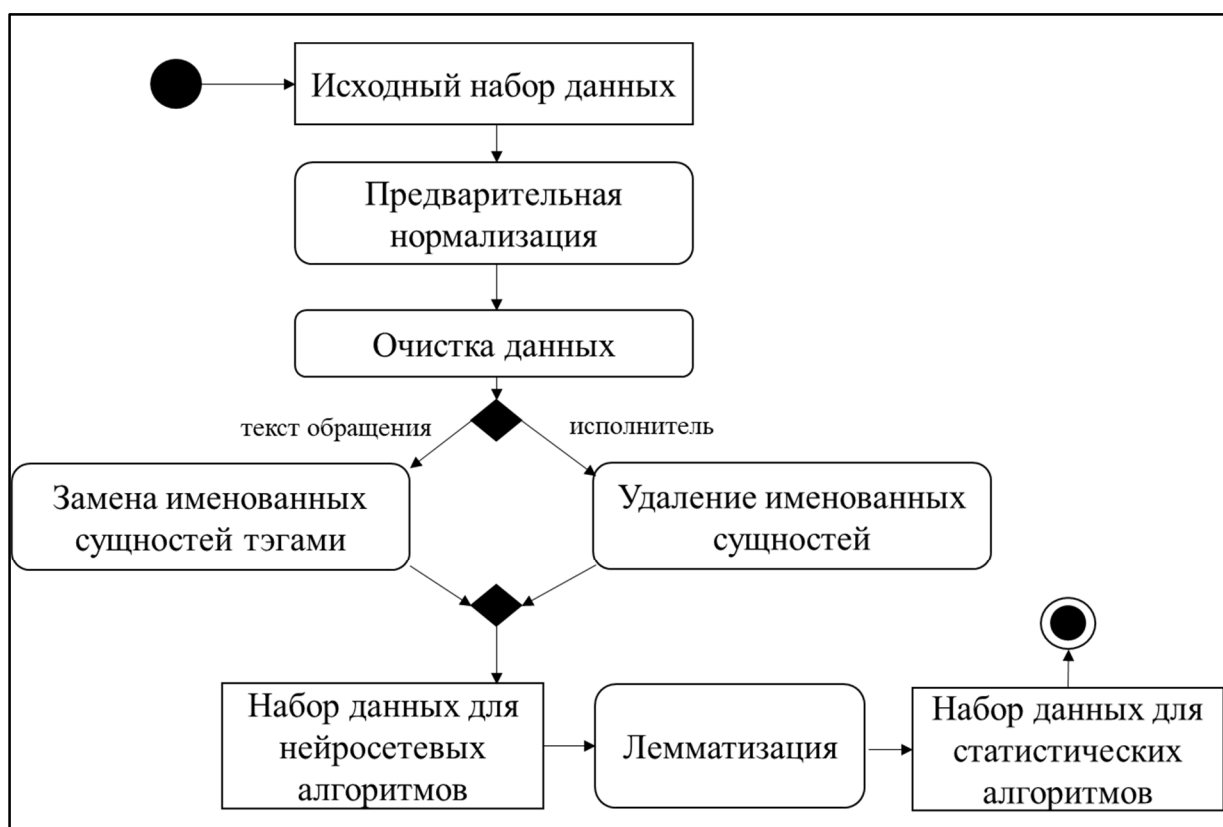


Рисунок 30 – Алгоритм построения модели BigARTM

4.4. Реализация микросервиса на архитектуре REST

REST (Representational State Transfer) – это SOA (сервис-ориентированная архитектура) и архитектурный стиль для распределенных гипермедиа-систем, использующих протокол HTTP для связи. Для реализации API микросервиса на архитектуре REST был использован фреймворк Flask.

При запуске сервера инициализируются все библиотеки и в оперативную память загружаются веса предобученной модели BERT. Инициализируется база данных (рисунок 10).

Анализ обращения выполняется сервером в несколько этапов:

- 1) выполняется предобработка сообщения;
- 2) выполняется предсказание топ-3 категорий моделью BERT;
- 3) для наиболее вероятной категории загружается модель BigARTM и находится топ-3 тематик обращения;
- 4) для наиболее вероятных категории и темы загружается топ-3 исполнителей.

Для взаимодействия с сервером клиент должен использовать следующие методы.

1. POST /message. Это основной метод, который используется для передачи текста обращения на сервер и выдачи результатов анализа клиенту. Пользователь отправляет запрос серверу с параметрами login и password, которые используются для идентификации пользователя на сервере. В body запроса клиент отправляет текст в формате JSON с ключом message. Клиент получает ответ в формате JSON с ключами result, top1_category, category, subcategory, executor, top_words, message. JSON содержит результаты классификации и служебную информацию о статусе операции. Код функции обработки данного метода представлен на рисунке 31.

```

@app.route('/message', methods=['POST'])
#загрузка новых обращений граждан в систему и выдача результата
def message_post_request():
    check = check_username()
    if not check:
        return json.dumps({'result': \
'Некорректный логин или пароль!'}).encode('utf-8')
    try:
        user = request.args.get('username')
        body = json.loads(request.data.decode('utf-8'))
        message = body['mess']
        #запуск bert
        mess_bert = preprocessing_for_bert(message)
        predict_values = bert_predict(mess_bert)
        top3_values = bert_top3(predict_values)
        top1_category = top3_values.loc[top3_values['value'].idxmax(), \
'category']
        cat_dict = top3_values.to_dict()
        #запуск bigartm
        mess_bigartm = preprocessing_for_bigartm(mess_bert)
        predict_values_artm = bigartm_predict(mess_bigartm, top1_category)
        top3_artm = bigartm_top3(predict_values_artm, top1_category)
        top_words = top3_artm.loc[top3_artm['value'].idxmax(), \
'top_words']
        top1_sbj = top3_artm.loc[top3_artm['value'].idxmax(), 'sbj']
        subcat_dict = top3_artm.to_dict()
        #вывод исполнителей
        executor_dict = executor_top3(top1_category, top1_sbj)
        #сохранить обращение в базу данных
        store_message(user, message, cat_dict, subcat_dict, \
executor_dict, top_words)
    except Exception:
        print('Ошибка обработки обращения')
        return json.dumps({'result': 'ERROR'})
    return json.dumps({'result': 'SUCCESS', 'top1_category': \
top1_category, 'category': cat_dict, 'subcategory': subcat_dict, \
'executor': executor_dict, 'top_words': top_words, \
'message':body}).encode('utf-8')

```

Рисунок 31 – Алгоритм анализа сообщения пользователя на сервере

При получении запроса от клиента, сервер проверяет его имя пользователя и пароль на соответствие хранящимся на сервере. Затем, если полученные данные верны, выполняется предобработка переданного сообщения в соответствии с 4.2. Если на данном этапе выявлены ошибки, обрабатывается исключение и сервер отправляет клиенту JSON с сообщением ERROR. В случае корректной предобработки данных, выполняется предсказание для моделей BERT и BigARTM. Далее все полученные данные сохраняются в базе данных SQLite3, результат отправляется клиенту.

2. GET /message. Метод позволяет получить заданное число последних запросов для данного пользователя с полным их анализом, выводом

даты запроса и имени пользователя. Администратор может получить информацию о запросах всех пользователей, даже если пользователь скрыл свою историю запросов. Код данного метода приведен на рисунке 32.

```
def get_message(username, count):
    try:
        conn = sqlite3.connect(os.path.join(currdir, r"flask/log.db"))
        with conn:
            cursor = conn.cursor()
            create_table_if_not_exists(conn, cursor)
            if username != 'admin':
                query = 'SELECT * FROM mytable WHERE username = ? AND \
                    isvisible = "1" ORDER BY datetime DESC LIMIT ?'
                param_list = [username, count]
            else:
                query = 'SELECT * FROM mytable ORDER BY datetime DESC \
                    LIMIT ?'
                param_list = [count]
            cursor.execute(query, param_list)
            conn.commit()
            rows = cursor.fetchall()
            print(rows)
    except Exception:
        print('Ошибка обработки')
    return rows

@app.route('/message', methods=['GET'])
def message_get_request():
    check = check_username()
    if not check:
        return json.dumps({'result': \
            'Некорректный логин или пароль!'}).encode('utf-8')
    username = request.args.get('username')
    count = request.args.get('count')
    mess_list = get_message(username, count)
    return json.dumps({'result': mess_list}).encode('utf-8')
```

Рисунок 32 – Алгоритм вывода истории пользователей

3. DELETE /message. Метод позволяет пометить все обращения, загруженные пользователем, недоступными к просмотру. Если в качестве параметров данного метода передаются логин и пароль администратора, то очищается история всех пользователей.

4. GET /statistics. Метод используется для получения статистики запросов пользователя. Если метод вызывает администратор, метод возвращает статистику запросов по всем пользователям системы. Релизация метода аналогична представленному на рисунке 31, за исключением запроса к базе данных и структуры JSON ответа.

5. GET /categories. Возвращает топ-3 категорий последнего запроса пользователя, формат выходных данных аналогичен запросу POST.

6. GET /topwords. Возвращает 10 ключевых слов темы последнего запроса пользователя, формат выходных данных аналогичен запросу POST.

7. GET /executor. Возвращает топ-3 исполнителей последнего запроса пользователя, формат выходных данных аналогичен запросу POST.

8. POST /newuser. Вносит нового пользователя в список пользователей на сервере.

По умолчанию сервер доступен на localhost, порт 5000.

4.5. Реализация интерфейса пользователя

Пользовательский интерфейс клиента реализован на фреймворке Streamlit. Клиент обменивается данными с сервером по API, описанному в 4.4.

При запуске веб-интерфейса (по умолчанию localhost:8501) в браузере отображаются элементы пользовательского интерфейса, которые приведены на рисунке 33.

Перед началом работы в системе пользователь должен авторизоваться, введя свои логин и пароль. Авторизованный пользователь может вводить текст сообщения и получать результаты его анализа. По умолчанию пользователю предлагается логин test и пароль test.

После ввода текста обращения и нажатия кнопки «Анализировать» пользователю предоставляются результаты анализа введенного текста. Результаты включают в себя сам текст обращения, наиболее вероятную категорию обращения, а также топ-3 наиболее вероятных категорий, топ-3 тем и топ-3 исполнителей.

Для каждой таблицы с результатами предусмотрена визуализация в виде столбчатой диаграммы, реализованная с помощью библиотеки altair.

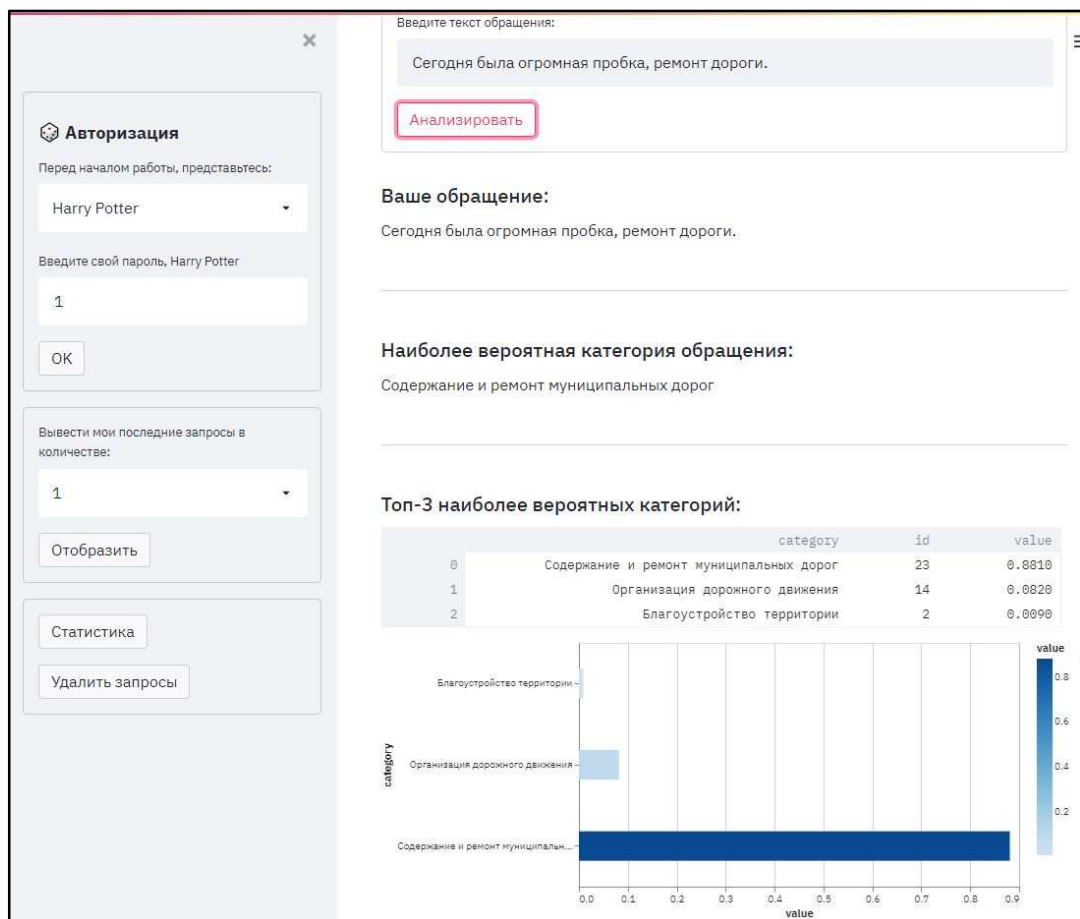


Рисунок 33 – Элементы пользовательского интерфейса

На рисунке 34 приведен вывод информации пользователю о топ-3 тем введенного обращения.

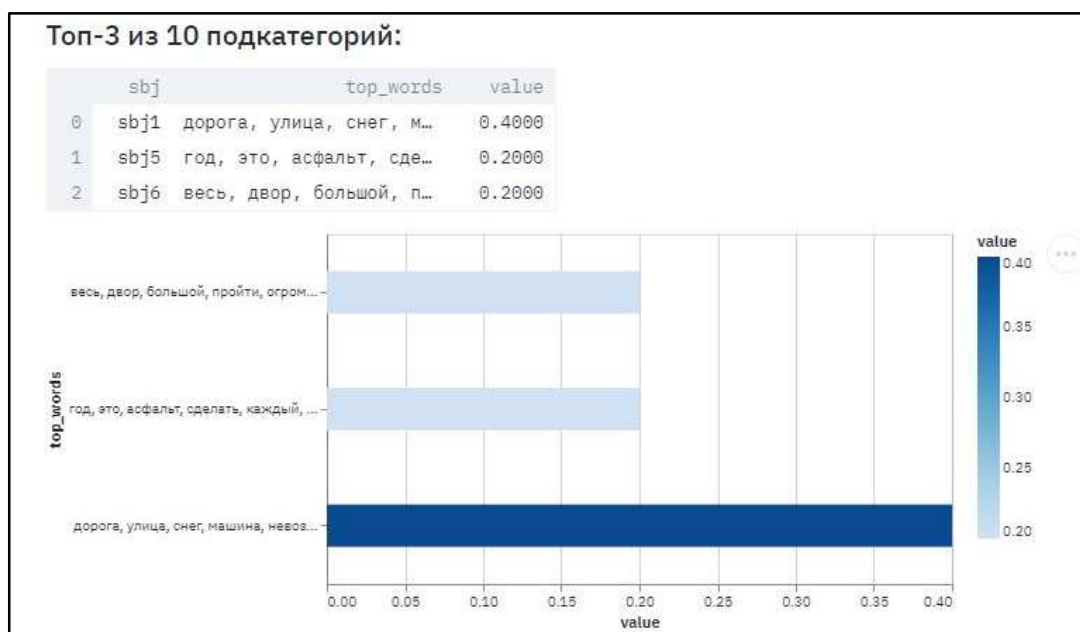


Рисунок 34 – Топ-3 тем введенного обращения

На рисунке 35 приведен вывод информации пользователю о топ-3 наиболее вероятных исполнителях.

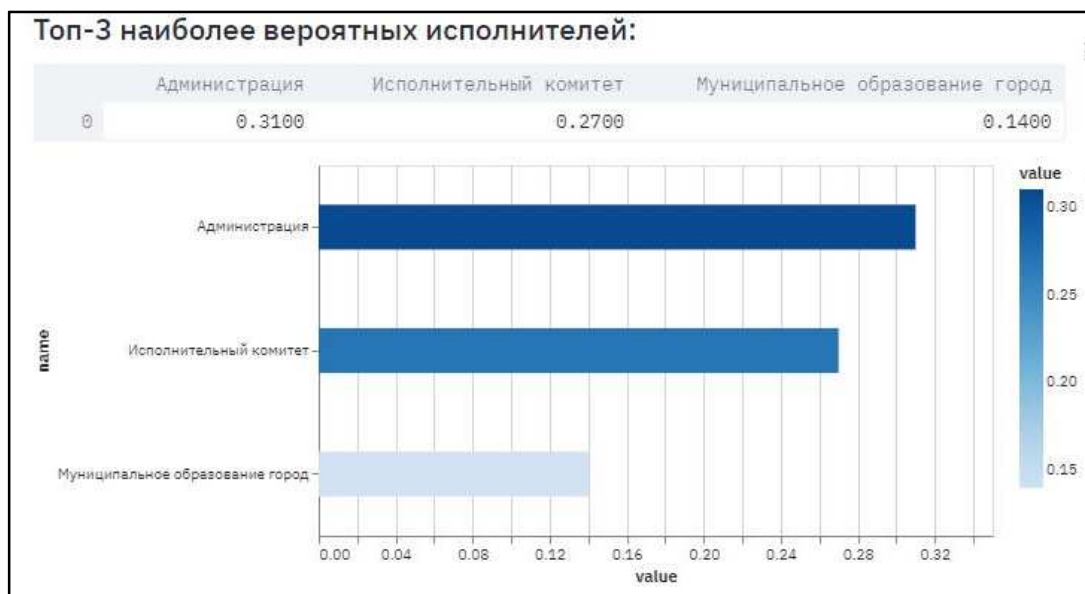


Рисунок 35 – Топ-3 наиболее вероятных исполнителей

На рисунке 36 приведен вывод статистики запросов пользователей администратору системы.

Авторизация

Перед началом работы, представьтесь:

admin

Введите свой пароль, admin

admin

ОК

Вывести мои последние запросы в количестве:

1

Отобразить

Статистика

Удалить запросы

Введите текст обращения:

Анализировать

Статистика запросов пользователей:

	Albus Dumbledore	Harry Potter	admin
Число запросов	1	3	1

Рисунок 36 – Статистика запросов пользователей

Развертывание системы с помощью контейнеров docker

На данный момент проект находится в открытом доступе на <https://hub.docker.com/>. Для загрузки и запуска проекта файл `docker-compose.yml` необходимо загрузить на локальную машину с установленным `docker`.

Команда `docker-compose up` развертывает 2 контейнера, один из которых представляет собой сервер на базе фреймворка Flask, а во втором контейнере находится клиент с пользовательским веб-интерфейсом на базе фреймворка Streamlit.

Доступ к пользовательскому интерфейсу можно получить по `http://localhost:80`. Для доступа к системе необходимо знать пароль. Пароль и логин администратора: `admin`, `admin`. Доступ к серверу можно получить с помощью REST API на `http://localhost:5000`.

На рисунке 37 приведено содержимое файла `Dockerfile` для сборки `docker`-образа сервера.

```
FROM python:3.6-stretch
MAINTAINER vitomskov <vitomskov@mail.ru>
# устанавливаем параметры сборки
RUN apt-get update && \
apt-get install -y gcc \
make apt-transport-https ca-certificates build-essential
# проверяем окружение python
RUN python3 --version
RUN pip3 --version
# set a directory for the app
WORKDIR /usr/src/message2category/src
# install dependencies
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
# copy all the files to the container
COPY /src/ .
# tell the port number the container should expose
EXPOSE 5000
# run the command
CMD ["python", "docker_flask.py"]
```

Рисунок 37 – Сборка `docker`-образа сервера

На рисунке 38 приведено содержимое файла `docker-compose.yml` для развертывания системы с помощью утилиты `docker-compose`.

```
version: "3"
services:
  message2category:
    image: aleks1212v/mess2category:1.2
    container_name: message2category
    environment:
      - discovery.type=single-node
    ports:
      - 5000:5000
  shell_m2c:
    image: aleks1212v/shell_m2c:1.2
    container_name: shellm2c
    depends_on:
      - message2category
    ports:
      - 80:8501
```

Рисунок 38 – Развертывание системы с помощью docker-compose

Данная система была развернута в облаке AWS EC2 и доступна для тестирования. URL: <http://ec2-3-237-9-85.compute-1.amazonaws.com/>

Выводы по разделу 4

Можно подвести следующие итоги реализации системы.

1. Разработан инструмент предобработки данных с целью подготовки набора данных для обучения моделей. В обработке данных мной были использованы регулярные выражения (библиотека `re`), нахождение именованных сущностей (библиотека `natasha`) и лемматизация слов с целью сокращения словаря статистических моделей (библиотека `rumorphy2`).

2. Исследованы статистические и нейросетевые модели для определения категории обращений граждан из набора данных республики Татарстан. В итоге наилучшие результаты на тестовых данных показала нейросетевая модель BERT sentence_ru реализации DeepPavlov. Значение метрик для данной модели составило $F1 = 84,5\%$, $MCC = 0,789$. Данная модель была в итоге использована в системе. Полученные результаты могут быть в дальнейшем улучшены путем дообучения модели на новых данных.

3. Исследована иерархическая тематическая модель на основе аддитивной регуляризации для определения темы обращения (библиотека

BigARTM). Для каждой категории были определены от 3 до 10 тем, в зависимости от числа обращений. В итоге для каждой категории из набора данных получен файл с весами BigARTM-модели, файл со списком топ-10 слов каждой из тем и файл с топ-3 наиболее вероятных исполнителей каждой из тем.

4. Реализован REST-сервер на фреймворке Flask, который позволяет оперативно анализировать новые обращения граждан, хранить и выводить историю обращений и результаты их анализа, а также поддерживает многопользовательский режим работы.

5. Разработан клиент, реализующий пользовательский веб-интерфейс на фреймворке Streamlit.

6. Выполнена контейнеризация сервера и клиента. Docker-образы доступны на dockerhub. Сервер: `aleks1212v/mess2category:1.2`; клиент: `aleks1212v/shell_m2c:1.2`.

7. Выполнена автоматизация развертывания контейнеров сервера и клиента с помощью утилиты `docker-compose`.

8. Система развернута в облаке AWS EC2 и доступна для тестирования. URL: <http://ec2-3-237-9-85.compute-1.amazonaws.com>.

При реализации проекта были использованы:

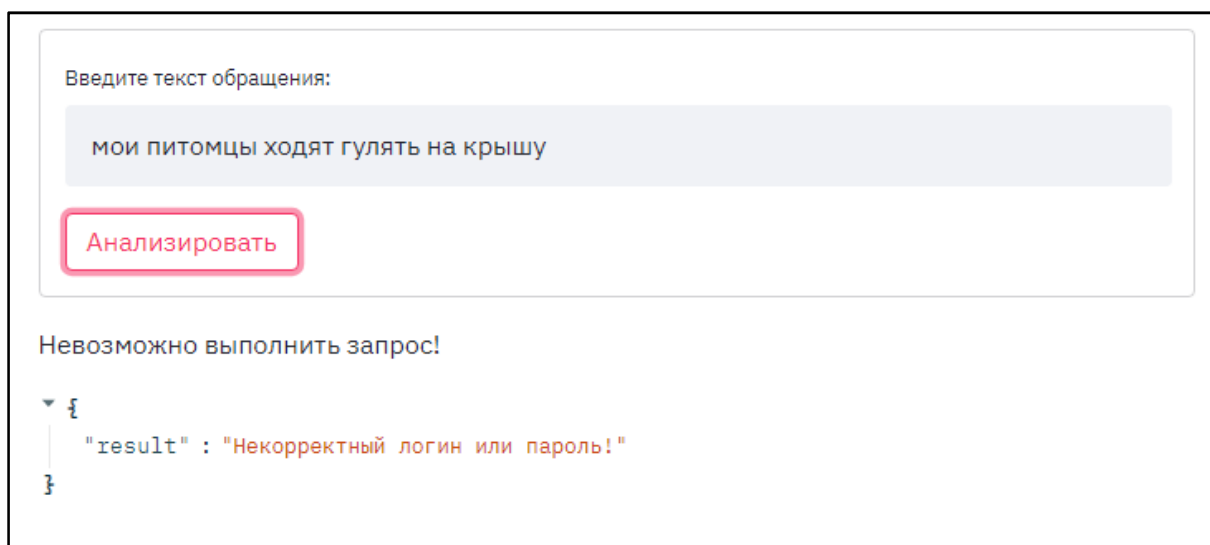
- 1) платформа Paperspace Gradient для обучения моделей и отладки кода;
- 2) локальный компьютер с ОС Windows 10 и установленным docker для создания docker-образов;
- 3) платформа AWS EC2 для развертывания системы.

5. ТЕСТИРОВАНИЕ

Тестирование веб-сервиса

Для тестирования использовался веб-сервис, запущенный в контейнере docker. Были произведены попытки ввода некорректных сообщений – пустых или состоящих из нечитаемых сочетаний символов. Отдельно были проведены тесты сообщений, набранных на английской раскладке, которые модель RuBERT обрабатывать не может. Во всех случаях сервером корректно обрабатывалось исключение, и пользователю выдавалось соответствующее диагностическое сообщение.

Были проведены тесты неудачной идентификации пользователя, а также проанализированы результаты обработки нажатий кнопок интерфейса в сочетании с различным наполнением базы данных системы. На рисунке 39 приведен пример вывода диагностического сообщения пользователю при неправильном вводе пароля.



Введите текст обращения:

мои питомцы ходят гулять на крышу

Анализировать

Невозможно выполнить запрос!

```
{
  "result" : "Некорректный логин или пароль!"
}
```

Рисунок 39 – Диагностическое сообщение пользователю

Во всех случаях наблюдалась устойчивая работа системы, время отклика не превышало 1 секунды.

Тестирование алгоритмов анализа текстовых данных

Для тестирования алгоритмов анализа текстовых данных использовался REST-сервер, запущенный в контейнере docker.

Одной из поставленных передо мной задач стояла задача классификации обращений по исполнителям. Исполнители могут обрабатывать сразу несколько подкатегорий или категорий обращений, а одни и те же обращения могут обрабатываться разными исполнителями.

В рамках тестирования алгоритмов анализа текстовых данных были использованы метрики качества предсказания системой исполнителей. В исследовании был использован весь исходный набор данных. Вычислялись средневзвешенные значения метрик `f1_score`, `recall_score`, `precision_score`, `accuracy_score`. В данном случае важность классов пропорциональна количеству образцов, т. е. класс, который недостаточно представлен, считается менее важным.

Для тестирования использовался REST API сервера. В систему по одному передавались тексты всех обращений исходного набора данных.

Если истинная метка исполнителя была первой в топ-3 исполнителей (либо второй или третьей, но при условии, что вероятности равны), то данная метка добавлялась в списки №1, №2 и №3.

Если метка была второй (или третьей, но при условии, что вероятности равны), то она добавлялась только в списки №2 и №3, а в список №1 добавлялась метка, имеющая максимальную предсказанную вероятность.

Если метка была третьей, то она добавлялась только в №3, а в списки №1 и №2 добавлялась метка с максимальной предсказанной вероятностью. Если же метка вообще не попала в топ-3, то во все списки добавлялись метки, имеющие максимальную предсказанную вероятность.

Код тестирования системы приведен на рисунке 40.

```

for message, label in zip(data['description'], data['worker']):
    if raw_json['result'] != 'SUCCESS' or raw_json == []:
        Y_pred_top1.append(executor_names.index('Неизвестный исп.'))
        Y_pred_top2.append(executor_names.index('Неизвестный исп.'))
        Y_pred_top3.append(executor_names.index('Неизвестный исп.'))
        continue
    else:
        executor = pd.DataFrame([raw_json['executor']])
        ex = executor.T
        ex.columns = ['value']
        ex['name'] = ex.index
        if (ex['name'].shape[0] > 0 and ex['name'][0] == label) or \
            (ex['name'].shape[0] > 1 and ex['name'][1] == label and \
             ex['value'][1] == ex['value'][0]) or \
            (ex['name'].shape[0] > 2 and ex['name'][2] == label and \
             ex['value'][2] == ex['value'][0]):
            #элемент в топ-1
            Y_pred_top1.append(executor_names.index(label))
            Y_pred_top2.append(executor_names.index(label))
            Y_pred_top3.append(executor_names.index(label))
        elif (ex['name'].shape[0] > 1 and ex['name'][1] == label) or \
            (ex['name'].shape[0] > 2 and ex['name'][2] == label and \
             ex['value'][2] == ex['value'][1]):
            #элемент в топ-2
            Y_pred_top1.append(executor_names.index(ex['name'][0]))
            Y_pred_top2.append(executor_names.index(label))
            Y_pred_top3.append(executor_names.index(label))
        elif (ex['name'].shape[0] > 2 and ex['name'][2] == label):
            #элемент в топ-3
            Y_pred_top1.append(executor_names.index(ex['name'][0]))
            Y_pred_top2.append(executor_names.index(ex['name'][0]))
            Y_pred_top3.append(executor_names.index(label))
        else:
            #элемент не в топ-3
            Y_pred_top1.append(executor_names.index(ex['name'][0]))
            Y_pred_top2.append(executor_names.index(ex['name'][0]))
            Y_pred_top3.append(executor_names.index(ex['name'][0]))

```

Рисунок 40 – Расчет метрик качества для определения исполнителей

Таким образом, были рассчитаны метрики качества попадания исполнителя в список топ-1, топ-2 и топ-3. Значения метрик качества приведены в таблице 9.

Таблица 9 – Попадание исполнителей в топ-1, топ-2 и топ-3

Метрика	Топ-1	Топ-2	Топ-3
Accuracy	0.316	0.543	0.672
Precision (weighted)	0.348	0.498	0.640
Recall (weighted)	0.316	0.543	0.672
F1 (weighted)	0.185	0.417	0.568

Выводы по разделу 5

В процессе тестирования веб-сервиса система продемонстрировала стабильную работу и корректную обработку ошибок при различных действиях пользователя.

При тестировании алгоритмов анализа текстовых данных были получены метрики качества попадания исполнителя из поля «Последний исполнитель» исходного набора данных в топ-1, топ2- и топ-3 исполнителей, предсказанных системой. Достигнуты неплохие показатели.

Для дальнейшего роста качества определения исполнителей необходимы данные других регионов, так как исполнители в зависимости от региона будут отличаться. Требуется информация по всем исполнителям, а не только по последнему исполнителю, как в исходном наборе данных. Кроме того, для более точного анализа необходимо распределение исполнителей по группам (классам).

В процессе тестирования установлено, что разработанная система выполняет свои функции в требуемом объеме и может быть рекомендована для внедрения в регионах.

ЗАКЛЮЧЕНИЕ

В ходе данной работы была разработана система автоматизированной обработки обращений граждан.

В ходе работы были достигнуты поставленные цели:

- 1) проведен анализ существующих решений и методов автоматизированной обработки обращений граждан;
- 2) спроектирована архитектура системы;
- 3) выполнена предварительная подготовка данных;
- 4) выполнено тестирование статистических и нейросетевых моделей и отобрана лучшая;
- 5) разработан микросервис на архитектуре REST, а также веб-интерфейс пользователя системы;
- 6) проведено тестирование реализованной системы.

Данная работа была выполнена по инициативе заказчика – компании «Интерсвязь». По результату работы имеется акт о внедрении данного проекта в компании «Интерсвязь».

Выполненная работа является на сегодняшний день актуальной. В дальнейшем могут быть существенно улучшены метрики качества классификации обращений граждан путем обучения полученных моделей на новых данных; могут быть добавлены новые категории обращений граждан и новые исполнители.

ЛИТЕРАТУРА

1. Отчет о результатах рассмотрения обращений граждан за 2020 год. [Электронный ресурс] URL: https://mon.tatarstan.ru/statisticheskiedannye-o-rabote-s-obrashcheniyami.htm?pub_id=2639870 (дата обращения: 26.05.2021 г.).
2. Антонов С.В. Анализ практического применения подсистемы обработки текстовых сообщений. // Технические науки, 2016. - №4 (25). – С. 36–40.
3. Кварацхелия А.Г., Рахимов Д.Ф., Мангушева А.Р. Классификатор обращений граждан. [Электронный ресурс] URL: <https://git.asi.ru/damir.rakhimov/classification-appeals-ada> (дата обращения: 26.05.2021 г.).
4. Mikolov T., Chen K., Corrado G., Dean J. Efficient Estimation of Word Representations in Vector Space. [Электронный ресурс] URL: <https://arxiv.org/pdf/1301.3781> (дата обращения: 09.05.2021 г.).
5. Kusner M., Sun Y., Kolkin N., Weinberger K. From Word Embeddings To Document Distances. [Электронный ресурс] URL: <http://mkusner.github.io/publications/WMD> (дата обращения: 09.05.2021 г.).
6. Перевалов А.М, Бобков С.А. Автоматический классификатор обращений граждан. [Электронный ресурс] URL: <https://git.asi.ru/perevalovA/appeals-webservice> (дата обращения: 26.05.2021 г.).
7. Лейн Х., Хапке Х., Ховард К. Обработка естественного языка в действии. – СПб: Питер, 2020. – 576 с.
8. Гольдберг Й. Нейросетевые методы в обработке естественного языка / пер. с англ. А.А. Слинкина. – М.: ДМК Пресс, 2019. – 282 с.
9. Sennrich R., Haddow B., Birch A. Neural Machine Translation of Rare Words with Subword Units. [Электронный ресурс] URL: <https://arxiv.org/pdf/1508.07909> (дата обращения: 09.05.2021 г.).
10. Schuster M., Nakajima K. Japanese and Korean Voice Search. [Электронный ресурс] URL: <https://static.googleusercontent.com/media/research.google.com/ru/pubs/archive/37842> (дата обращения: 09.05.2021 г.).

11. Kudo T. Subword Regularization: Improving Neural Network Translation Models with Multiple Subword Candidates. [Электронный ресурс] URL: <https://arxiv.org/pdf/1804.10959> (дата обращения: 09.05.2021 г.).
12. Бенгфорт Б., Билбро Р., Охеда Т. Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. – СПб: Питер, 2019. – 368 с.
13. Пойнтер Я. Программируем с PyTorch. Создание приложений глубокого обучения. – СПб: Питер, 2020. – 256 с.
14. Vaswani A. et al. Attention is all you need. [Электронный ресурс] URL: <https://arxiv.org/abs/1706.03762> (дата обращения: 26.05.2021 г.).
15. Kuratov Y., Arkhipov M. Adaptation of deep bidirectional multilingual transformers for russian language. [Электронный ресурс] URL: <https://arxiv.org/pdf/1905.07213> (дата обращения: 09.05.2021 г.).
16. Arkhipov M., Trofimova M., Kuratov Y., Sorokin A. Tuning Multilingual Transformers for Named Entity Recognition on Slavic Languages. [Электронный ресурс] URL: <https://www.aclweb.org/anthology/W19-3712> (дата обращения: 09.05.2021 г.).
17. Vorontsov K. BigARTM Documentation. [Электронный ресурс] URL: https://bigartm.readthedocs.io/_/downloads/en/stable/pdf/ (дата обращения: 09.05.2021 г.).
18. Belyu A. V., Seleznova M. S., Sholokhov A. K., Vorontsov K. V. Quality Evaluation and Improvement for Hierarchical Topic Modeling // Computational Linguistics and Intellectual Technologies. Dialogue 2018. Pp. 110-123.
19. Русскоязычный корпус коротких текстов RuTweetCorp. [Электронный ресурс] URL: <https://study.mokoron.com> (дата обращения: 17.05.2021 г.).
20. Макмахан Б., Рао Д. Знакомство с PyTorch: глубокое обучение при обработке естественного языка. – СПб: Питер, 2020. – 256 с.

21. Предобученные модели BERT transformers. [Электронный ресурс] URL: https://huggingface.co/transformers/pretrained_models.html (дата обращения: 17.05.2021 г.).
22. Предобученные модели BERT deeppavlov. [Электронный ресурс] URL: <http://docs.deeppavlov.ai/en/master/features/models/bert.html> (дата обращения: 17.05.2021 г.).
23. Loshchilov I., Hutter F. Decoupled weight decay regularization. [Электронный ресурс] URL: <https://arxiv.org/abs/1711.05101> (дата обращения: 17.05.2021 г.).