

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Рязанский государственный радиотехнический университет имени В.Ф. Уткина»

«К защите»
Заведующий кафедрой ВПМ

_____ Овечкин Г.В.
«__» _____ 2023 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (магистратура)

на тему

**«Разработка программного обеспечения для исследования влияния
предобработки текста на точность решения задач классификации
текстовой информации»**

Направление подготовки: 09.04.04 Программная инженерия

Наименование ОПОП: Разработка программно-информационных систем

Руководитель ОПОП _____ (Овечкин Г.В.)

Руководитель _____ (Цуканова Н.И.)

Обучающийся _____ (Головкин Н.В.)

Рязань 2023

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Рязанский государственный радиотехнический университет
имени В.Ф. Уткина»

«УТВЕРЖДАЮ»

Заведующий кафедрой ВПМ

_____ Овечкин Г.В.

«06» апреля 2023 г.

ЗАДАНИЕ
на выпускную квалификационную работу

Обучающемуся Головкину Никите Вячеславовичу, студенту группы 143М
(фамилия, имя, отчество, № группы)

1. Тема ВКР Разработка программного обеспечения для исследования влияния предобработки текста на точность решения задач классификации текстовой информации

2. Срок представления законченной ВКР к защите: «5» июня 2023 г.

3. Руководитель Цуканова Нина Ивановна, РГРТУ, доцент
(фамилия, имя, отчество полностью, место работы, должность)

4. Исходные данные к ВКР Keras 2.12.0

Python 3.8

Django 4.2.0

Google Colab

5. Содержание ВКР

Введение

Постановка задачи

Технико-экономическое обоснование

Обоснование выбора средств разработки

Теоретическая часть

Практическая часть

Программная документация

Тестирование

Заключение

Список использованных источников

Приложение. Листинг наиболее значимых частей программы

6. Перечень графического (демонстрационного) материала

Постановка задачи

Разработка алгоритмов
Интерфейс программной системы
Тестирование программной системы

Дата выдачи задания: «6» апреля 2023 г.

Руководитель ОПОП _____
(подпись)

Руководитель _____
(подпись)

Задание принял к исполнению «6» апреля 2023 г.

Обучающийся _____
(подпись)

АННОТАЦИЯ

Головкин Никита Вячеславович

Направление подготовки: 09.04.04 Программная инженерия

ОПОП: Разработка программно-информационных систем

Тема выпускной квалификационной работы: Разработка программного обеспечения для исследования влияния предобработки текста на точность решения задач классификации текстовой информации

Руководитель ВКР: Цуканова Нина Ивановна, доцент, к.т.н., доцент каф. ВПИМ

(Фамилия Имя Отчество, ученая степень, ученое звание)

Выпускная квалификационная работа содержит:

... страниц, ... таблиц, ... рисунков, ... источников, ... приложений.

Объектом исследования является влияние предобработки текста на точность решения задач классификации текстовой информации.

Целью выпускной работы является повышение точности результатов, получаемых при классификации текстовых данных с использованием нейронных сетей при помощи применения различных методов предобработки текстовых данных.

В работе использованы методы предобработки текста, машинного обучения.

Научная новизна выпускной квалификационной работы заключается в исследовании влияния методов предобработки данных на точность классификации текстов на русском языке с помощью нейронных сетей.

При выполнении выпускной квалификационной работы получены следующие результаты: проведен анализ предметной области, рассмотрены аналоги, спроектировано, разработано и протестировано программное обеспечение, составлена программная документация.

Рекомендации по внедрению результатов или итоги внедрения: программное обеспечение готово к внедрению.

Область применения результатов исследования: классификация текстовых данных на русском языке.

Результаты исследования опубликованы в 6 статьях,

Цуканова Н.И., Александров В.В., Головкин Н.В., Шурыгина О.В. ОЦЕНКА КАЧЕСТВА КОЛЛЕКТИВНО-ДОГОВОРНЫХ АКТОВ В СФЕРЕ ОБРАЗОВАНИЯ С ПОМОЩЬЮ НЕЙРОННЫХ СЕТЕЙ. СОВРЕМЕННЫЕ ТЕХНОЛОГИИ В НАУКЕ И ОБРАЗОВАНИИ СНО-2023 VI

МЕЖДУНАРОДНЫЙ НАУЧНО-ТЕХНИЧЕСКИЙ ФОРУМ Сборник трудов
Том 4

Александров В.В., Цуканова Н.И., Головкин Н.В., Шурыгина О.В. Исследование способов повышения точности классификации фрагментов Коллективного договора. Математическое и программное обеспечение вычислительных систем: Межвуз. сб. науч. тр. / Под ред. Г.В. Овечкина - Рязань: РГРТУ им. В.Ф. Уткина, январь, 2023 - 124 с.

Цуканова Н.И., Головкин Н.В. Влияние методов нормализации слов на точность классификации текста. Математическое и программное обеспечение вычислительных систем: Межвуз. сб. науч. тр. / Под ред. Г.В. Овечкина - Рязань: РГРТУ им. В.Ф. Уткина, 2022-124 с

Цуканова Н.И., Александров В.В., Головкин Н.В., Шурыгина О.В. Иерархическая классификация текстов с помощью нейронных сетей. Современные технологии в науке и образовании - СТНО-2022. Сборник трудов V Международного научно-технического форума: в 10 т. Т.4. / под общ. ред. О.В. Миловзорова. – Рязань: Рязан. гос. радиотехн. ун-т, 2022

Цуканова Н.И., Головкин Н.В. Влияние предобработки текста на точность многоклассовой классификации. МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ: Межвуз. сб. науч. тр. / Под ред. Г.В. Овечкина - Рязань: РГРТУ им. В.Ф. Уткина, январь 2023-124 с

Александров В.В., Цуканова Н.И., Головкин Н.В., Шурыгина О.В. Методы интеллектуального анализа данных и машинного обучения при автоматизированной оценке эффективности коллективных договоров. Межвузовский сборник науч. трудов "Математическое и программное обеспечение вычислительных систем", РГРТУ, 2022 г., с. 14-19

Работа выполнена на основе статистической информации, собранной в ведомственной лаборатории РГРТУ по автоматизированному контролю, анализу и оценки эффективности коллективно-договорных актов в сфере образования.

ABSTRACT

Golovkin Nikita Vyacheslavovich

Direction of training: 09.04.04 Software Engineering

BPEP: Development of software and information systems

Theme of the final qualifying work: Development of software for researching the effect of text preprocessing on the accuracy of text classification tasks

The head of final qualifying work: Tsukanova Nina Ivanovna, docent, Ph.D., docent of the Department of Computational and Applied Mathematics

Final qualifying work contains:

... pages, ... tables, ... figures, ... sources, ... attachments.

The object of the research is the effect of text preprocessing on the accuracy of text classification.

The aim of the final qualifying work is to improve the accuracy of the results obtained in the classification of text data using neural networks through the use of various methods of text data preprocessing.

The methods of text data preprocessing and machine learning were used in the work.

Scientific novelty of the final qualifying work is the research of the effect of preprocessing methods on the accuracy of the classification of texts in Russian with neural networks.

The results of the work – the analysis of the subject area, consideration of the analogues, designed, developed and tested software, compiled software documentation.

Recommendations for the implementation of the results or the results of the implementation: software ready for implementation.

Scope of the research results: classification of text data in Russian.

Research results published in 6 articles,

N.I. Tsukanova, V.V. Alexandrov, N.V. Golovkin, O.V. Shurygina. Assessing the Quality of Collective Agreement Acts in Education Using Neural Networks. Modern Technologies in Science and Education. International technical forum. Collection of works. Vol.4

Aleksandrov V.V., Tsukanova N.I., Golovkin N.V., Shurygina O.V. Research of methods to improve the accuracy of the classification of fragments of the Collective Agreement. Mathematical and software computing systems: Interuniversity collection

of scientific papers / Edited by G.V. Ovechkin - Ryazan: RSREU named after V.F. Utkin, January, 2023 - 124 p.

Tsukanova N.I., Golovkin N.V. The effect of word normalization methods on the accuracy of text classification. Mathematical and Software Computing Systems: Interuniversity collection of scientific papers / Edited by G.V. Ovechkin - Ryazan: V.F. Utkin RSREU, 2022-124 p.

Tsukanova N.I., Aleksandrov V.V., Golovkin N.V., Shurygina O.V. Hierarchical classification of texts using neural networks. Modern Technologies in Science and Education - STNO-2022. Proceedings of the V International Scientific and Technical Forum: in 10 vol. Under edition of O.V. Milovzorov. - Ryazan: Ryazan State Radio Engineering University, 2022

Tsukanova N.I., Golovkin N.V. The effect of text preprocessing on the accuracy of multiclass classification. MATHEMATICAL AND SOFTWARE COMPUTING SYSTEMS: Interuniversity collection of scientific works / Edited by G.V.Ovechkin - Ryazan: V.F.Utkin Russian State Radio Engineering University, January 2023-124 p.

Aleksandrov V. V., Tsukanova N. I., Golovkin N. V., Shurygina O. V. Methods of Intelligent Data Analysis and Machine Learning in Automated Assessment of Collective Agreements Effectiveness. Interuniversity Collection of Scientific Works "Mathematical and Software of Computing Systems", RGRTU, 2022, pp. 14-19

The work was performed on the basis of statistical information collected in the departmental laboratory of the Russian State Technical University for Automated Control, Analysis and Evaluation of the Effectiveness of Collective Agreements in the Field of Education.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	10
1 ПОСТАНОВКА ЗАДАЧИ.....	13
2 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ.....	15
3 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ.....	16
4 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ	18
4.1 Основные сведения о машинном обучении	18
4.2 Введение в искусственные нейронные сети	19
4.3 Предобработка текстового набора данных	24
4.3.1 Простые методы предобработки	24
4.3.2 Методы нормализации	27
4.3.3 Методы векторизации	29
4.3.4 Методы балансирования	35
4.4 Архитектуры нейронных сетей для классификации текста	38
4.4.1 Рекуррентные нейронные сети	39
4.4.2 Нейронные сети с долгой краткосрочной памятью	39
4.5 Решение задачи классификации	40
4.6 Оценка качества обучения	41
5 ПРАКТИЧЕСКАЯ ЧАСТЬ.....	44
5.1 Проектирование функциональности программного обеспечения	44
5.2 Разработка архитектуры программного обеспечения	46
5.3 Разработка модуля предобработки текстового набора данных	47
5.3.1 Описание входных и выходных данных	48
5.3.2 Разработка алгоритмов	48
5.3.3 Программная реализация	50
5.4 Разработка приложения для обучения моделей	55
5.4.1 Описание входных и выходных данных	55
5.4.2 Разработка алгоритмов	57

5.4.3 Программная реализация алгоритмов	57
5.5 Разработка веб-приложения	59
5.5.1 Разработка алгоритма	60
5.5.2 Разработка классов	61
5.5.3 Программная реализация	62
5.5.4 Разработка пользовательского интерфейса	63
6 ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ.....	66
6.1 Описание применения	66
6.1.1 Назначение программы	66
6.1.2 Условия применения	66
6.1.3 Описание задачи	68
6.1.4 Входные и выходные данные	68
6.2 Руководство оператора	70
6.2.1 Назначение программы	70
6.2.2 Условия выполнения программы	70
6.2.3 Выполнение программы	71
6.2.4 Сообщения оператору	78
6.3 Руководство программиста	79
6.3.1 Назначение и условия применения программы	79
6.3.2 Обращения к программе	80
6.3.3 Входные и выходные данные	81
6.3.4 Сообщения	81
6.4 Руководство системного программиста	81
6.4.1 Общие сведения о программе	81
6.4.2 Структура программы	82
6.4.3 Настройка программы	83
6.4.4 Проверка программы	83
6.4.5 Сообщения системному программисту	83

7 ТЕСТИРОВАНИЕ	84
7.1 Приложение для обучения моделей	88
7.1.1 План тестирования	88
7.1.2 Результаты тестирования	89
7.1.3 Анализ результатов тестирования	90
7.1.4 Исследование влияния предобработки текста на точность классификации текстовых данных	Ошибка! Закладка не определена.
7.2 Веб-приложение	90
7.2.1 План тестирования	90
7.2.2 Результаты тестирования	90
7.2.3 Анализ результатов тестирования	91
ЗАКЛЮЧЕНИЕ	92
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	93
ПРИЛОЖЕНИЕ А. ЛИСТИНГ НАИБОЛЕЕ ЗНАЧИМЫХ ЧАСТЕЙ ПРОГРАММЫ.....	94

ВВЕДЕНИЕ

В последние два десятилетия объем доступных текстовых данных заметно увеличился. Но от такого объема не будет пользы, если данные не будут организованы и проанализированы. Заниматься анализом огромных объемов текстовых данных вручную - это трудоемкая и практически невыполнимая задача. Однако благодаря достижениям в области машинного обучения и обработки естественного языка стало возможным быстро и эффективно структурировать и анализировать текстовые данные. Первым шагом в анализе таких данных является классификация текста.

Одним из инструментов анализа текстовых данных являются нейронные сети, однако, они работают лишь с числами, поэтому прежде чем подать данные на вход их необходимо предобработать. Существуют различные способы предобработки текстовых данных и каждый из них влияет на точность результатов, получаемых с помощью нейронных сетей.

Целью выпускной работы является повышение точности результатов, получаемых при классификации текстовых данных с использованием нейронных сетей при помощи применения различных методов предобработки текстовых данных.

Актуальность темы работы обусловлена ростом объема текстовых данных, которые необходимо проанализировать. На текущий момент анализ таких данных производится либо вручную, что является крайне неэффективным, либо с помощью использования интеллектуальных методов анализа данных, в том числе нейронных сетей, на точность которых можно повлиять с помощью применения различных методов предобработки текстовых данных.

Практическая значимость работы состоит в том, что создаваемое программное обеспечение позволит повысить точность классификации текстовых данных при использовании нейронных сетей.

Пояснительная записка к выпускной квалификационной работе содержит следующие основные разделы: «Постановка задачи», «Технико-экономическое

обоснование», «Обоснование выбора средств разработки», «Теоретическая часть», «Практическая часть», «Программная документация» и «Тестирование».

Раздел «Постановка задачи» включает в себя описание цели проекта, основных требований, предъявляемых к разрабатываемому программному обеспечению, а также перечень задач, которые необходимо решить для достижения поставленной цели.

В разделе «Технико-экономическое обоснование» приводится обоснование потребности в разработке программного обеспечения и анализ существующих программных продуктов, являющихся в той или иной степени аналогами разрабатываемого.

В разделе «Обоснование выбора средств» приводится обоснование выбранных программных средств для решения поставленных задач.

Раздел «Теоретическая часть» содержит сведения о различных методах предобработки текстовых данных, нейронных сетях, их обучении и оценке качества полученных моделей.

Раздел «Практическая часть» включает в себя проектирование функциональности и разработку архитектуры программного средства, а также саму разработку программного обеспечения.

В разделе «Программная документация» представлено руководство оператора, руководство программиста и руководство администратора. Рассмотрены такие вопросы, как назначение системы, требования к составу и параметрам технических средств, описана работа системы.

В разделе «Тестирование» описывается процесс тестирования разработанного программного средства, а также приводятся результаты исследования влияния различных методов предобработки текстовых данных на точность их классификации при помощи нейронных сетей.

В результате выполнения выпускной квалификационной работы было разработано программное обеспечение для предобработки и классификации текстовых данных, которое также позволило исследовать влияние различных

методов предобработки текстовых данных на точность их классификации при помощи нейронных сетей.

Разработка программного обеспечения велась на языке программирования *Python* (версия 3.8, среда разработки *PyCharm Community Edition*) с применением операционной системы Ubuntu 20.04.

В процессе выполнения выпускной квалификационной работы были получены следующие результаты:

- разработан модуль предобработки текстовых данных;
- разработано приложение для обучения моделей, позволяющее синтезировать модель нейронной сети для классификации текстовых данных;
- разработано веб-приложение, использующее синтезированную модель для классификации текстовых данных;
- произведено исследование влияния различных методов предобработки текстовых данных на точность их классификации при помощи нейронных сетей.

1 ПОСТАНОВКА ЗАДАЧИ

Как было сказано во введении, целью данной работы является повышение точности результатов, получаемых при классификации текстовых данных с использованием нейронных сетей при помощи применения различных методов предобработки текстовых данных.

Для достижения поставленной цели, необходимо разработать программное обеспечение, к которому предъявляются следующие требования:

- программное обеспечение должно позволять производить предобработку текстовых данных различными методами;
- программное средство должно иметь возможность синтезировать модель для классификации текста на основе уже имеющихся данных;
- каждый пользователь должен иметь возможность классифицировать текстовые данные при помощи синтезированной модели;
- программное средство для классификации текстовых данных должно обладать графическим интерфейсом.

Для выполнения предъявленных требований необходимо решить следующие задачи:

- определить архитектуру разрабатываемого программного обеспечения;
- рассмотреть различные методы предобработки текстовых данных;
- реализовать рассмотренные методы предобработки текстовых данных;
- выбрать программное средство для реализации модели;
- собрать данные, необходимые для обучения модели;
- синтезировать модели с использованием данных, предобработанных с помощью различных методов;
- произвести оценку качества полученных моделей;
- разработать приложение, которое бы позволяло использовать синтезированную модель для классификации текста;
- составить документацию к разработанному программному обеспечению;

- произвести тестирование разработанного программного обеспечения.

2 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ

На данный момент классификация текстовых данных производится либо вручную, либо с помощью интеллектуальных методов анализа данных. Ручной подход является трудоемким и неэффективным, как с точки зрения временных затрат, так и экономических. Интеллектуальные методы анализа данных лишены этих недостатков, но порой они могут обладать недостаточной точностью.

На точность результатов, получаемых с помощью интеллектуальных методов анализа данных, в частности, нейронных сетей, можно повлиять с помощью различных методов предобработки данных.

Следовательно, существует потребность в создании такого программного обеспечения, которое бы позволило производить предобработку текстовых данных различными методами.

В настоящее время существуют программные средства, позволяющие производить предобработку текстовых данных, среди которых можно выделить следующие: NLTK, PyMorphy2, spaCy, Gensim. Обзор данных программных средств приведен в таблице 1.

Таблица 1 – Обзор существующих программных средств для предобработки текстовых данных

Программный продукт	Векторизация текста	Нормализация слов	Поддержка русского языка	Балансирование набора данных
NLTK	+	+	+	-
PyMorphy2	-	+	+	-
spaCy	-	-	-	-
Gensim	+	-	-	-

Стоит отметить, что большинство из них сконцентрированы лишь на небольшом подмножестве методов предобработки.

Таким образом, очевидна необходимость создания программного обеспечения, которое бы объединяло в себе различные методы предобработки текстовых данных.

Для создания такого программного обеспечения нужны соответствующие средства разработки.

3 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ

В качестве среды для разработки модуля предобработки текста и приложения для обучения моделей был выбран Google Colab. Такой выбор объясняется тем, что нейронные сети требуют наличия высокопроизводительной видеокарты с большим объемом видеопамяти. Google Colab — это бесплатная интерактивная виртуальная среда от Google, позволяющая создавать ноутбуки Jupyter Notebook с кодом на языке программирования Python. Google Colab предоставляет доступ к высокопроизводительным графическим процессорам, что как раз таки и позволяет удовлетворить потребность в таком аппаратном обеспечении. В данной работе обучение моделей производилось с использованием видеокарты NVIDIA Tesla T4.

Создание модуля предобработки текстовых данных и приложения для обучения моделей производится с использованием языка Python 3.8 и библиотеки для глубокого обучения Keras 2.12.0. Такой выбор сделан из-за того, что Python является относительно простым языком, обладает огромным количеством прикладных библиотек для исследования данных и является де-факто стандартом в мире машинного обучения [12]. Keras представляет из себя библиотеку, предоставляющую высокоуровневый интерфейс для создания различных видов нейросетей. Данная библиотека основана на низкоуровневой библиотеке для машинного обучения TensorFlow от компании Google. Keras, посредством TensorFlow, задействует видеокарту для выполнения вычислений над тензорами с высокой производительностью. Те же вычисления, если их производить на центральном процессоре, займут приблизительно в 10 раз больше времени [19]. Ближайшим аналогом Keras является библиотека fast.ai, но она не предоставляет того уровня контроля над архитектурой, создаваемой нейросети, что Keras.

Разрабатываемое программное обеспечение для классификации текстовых данных должно быть доступно каждому пользователю, желающему классифицировать текст. В связи с этим требованием было решено создать веб-приложение. Такой подход имеет ряд преимуществ:

- актуальность используемой модели. Все пользователи используют самую актуальную модель для классификации текстовых данных. Пользователям нет необходимости обновлять ПО;
- отсутствие необходимости в приобретении мощного компьютера. Так как все вычисления, связанные с использованием модели, производятся на специально отведенных для этого серверах, то нет необходимости иметь мощное аппаратное обеспечение. Достаточно лишь такого, которое сможет открыть браузер.

В качестве языка программирования для разработки веб-приложения был выбран язык Python вместе с фреймворками Django и Keras. Такой выбор обосновывается тем, что большинство продвинутых моделей, обученных с помощью Keras можно использовать только лишь при помощи Keras. Так как Keras доступен лишь для Python, то имеет смысл разрабатывать веб-приложение с использованием Python. Фреймворк Django позволяет создавать веб-приложения с применением шаблона проектирования MVC, что позволяет управлять сложностью создаваемого веб-приложения, путем разделения представления от логики работы приложения.

4 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

4.1 Основные сведения о машинном обучении

Машинное обучение – это класс компьютерных алгоритмов, характерной чертой которых является не прямое решение задачи, а обучение за счёт применения решений множества сходных задач [10].

В классическом программировании, люди создают правила (программу) и данные, которые должны быть обработаны в соответствии с этими правилами, а затем получают ответы. При машинном обучении человек вводит данные, а также ответы, соответствующие этим данным, и на выходе получает правила, которые затем могут быть применены к новым данным для получения соответствующих ответов (рисунок 1).

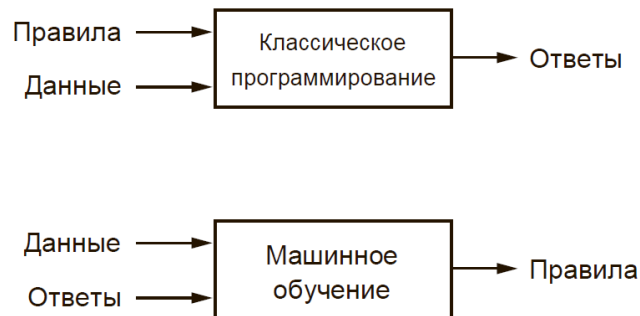


Рисунок 1 – Различия между классическим программированием и машинным обучением

Система машинного обучения обучается, а не программируется. Ей предъявляется множество примеров, относящихся к задаче, и она находит статистическую структуру в этих примерах, что в конечном итоге позволяет системе выработать правила для автоматизации задачи [10].

Процесс машинного обучения может быть разбит на следующие этапы подготовки и создания модели:

- подготовка данных (устранение дублирования, предобработка);
- выбор модели и ее обучение;
- оценка качества модели;
- использование модели для классификации новых примеров.

Рассмотрим более подробно такой подраздел машинного обучения, как искусственные нейронные сети.

4.2 Введение в искусственные нейронные сети

Искусственные нейронные сети – это математические модели, созданные по подобию биологических нейронных сетей. Они являются одним из методов машинного обучения [15].

Нейронная сеть основана на коллекции соединенных узлов, называемых искусственными нейронами. Каждое соединение, подобно синапсам в биологическом мозге, может передавать сигнал другим нейронам. Нейрон, получающий сигнал, обрабатывает его и может отправить другой сигнал соединенным с ним нейронам. Под сигналом понимается вещественное число. Значение на выходе каждого нейрона подсчитывается путем применения нелинейной функции к сумме его входов. Нелинейная функция используется для обеспечения более сложных взаимодействий. Соединения между нейронами называются гранями. У нейронов и граней есть веса, которые изменяются в процессе обучения. Веса увеличивают или уменьшают силу сигнала. Нейроны собраны в слои. Разные слои могут выполнять различные преобразования к значениям, поступающим на вход. Сигналы перемещаются от первого (входного) слоя к последнему (выходному) слою. Ниже на рисунке 2 приведено схематическое представление полносвязной сети – сети, в которой каждый нейрон предыдущего слоя связан с каждым нейроном следующего.

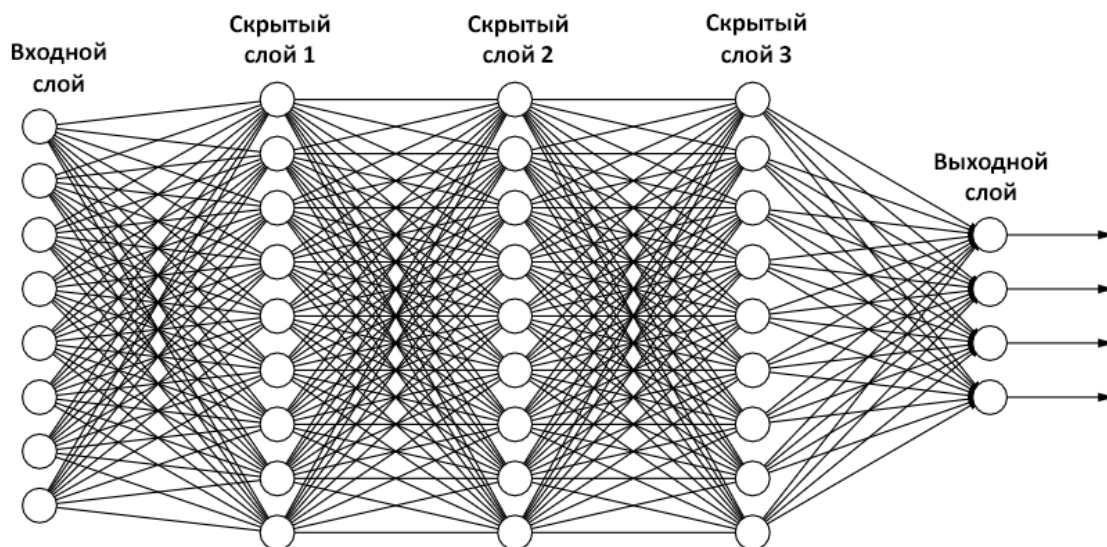


Рисунок 2 – Схематическое представление полносвязной сети

Нейронные сети обучаются путем обработки примеров, состоящих из входного значения и результата. Обучение нейронной сети обычно производится определением разницы между значением на выходе нейронной сети (предсказанием) и результатом из примера. Такая разница называется значением потерь. После этого сеть изменяет свои веса таким образом, чтобы уменьшить это значение. С каждой новой итерацией значение потерь становится все меньше и меньше, и, достигнув определенного критерия (например, точность превысила определенный порог), обучение нейронной сети приостанавливается [10].

При обучении нейронной сети выполняется прямой и обратный ход.

Прямой ход.

1. На входной слой нейронной сети подается тензор и распространяется по всей сети от слоя к слою.
2. Вычисляется выход сети.

Обратный ход.

1. Вычисляется разность между желаемым выходом сети и фактическим. В результате получается значение потерь.
2. Полученный сигнал распространяется в обратном направлении соединений с нейронами, и впоследствии корректируются веса сети с целью минимизации ошибки.

Так как нейронной сети нужно очень большое количество данных для обучения, уместить которое все сразу бывает чаще всего невозможно в ОЗУ компьютера или видеопамати видеокарты, то используют обучение при помощи батчей. Батч – это малая часть набора данных, которая подается на вход нейронной сети.

Обучение с помощью батчей происходит следующим образом.

1. Весь набор данных делится на определенное количество батчей.
2. На вход нейронной сети поступает батч, происходит прямой и обратный ход.
3. На вход нейронной сети поступает следующий батч, происходит прямой и обратный ход и т.д., пока на вход не поступят все батчи.

Когда на вход нейронной сети поступили все батчи и для каждого из них произошли прямой ход и обратный ход, то говорят, что прошла 1 эпоха.

В основе обучения нейронных сетей лежит применение стохастического градиентного спуска.

Если функция дифференцируема, то теоретически возможно найти ее минимум аналитически: известно, что минимум функции – это точка, где производная равна 0, поэтому достаточно найти все точки, где производная обращается в 0, и проверить, в какой из этих точек функция имеет наименьшее значение. Применительно к нейронной сети это означает аналитическое нахождение комбинации значений весов, которая дает наименьшее значение для функции потерь [10]. Этого можно достичь, решив уравнение (1) для W .

$$\nabla(f)(W) = 0, \quad (1)$$

где $\nabla(f)(W)$ – градиент функции f , f – функция нейронной сети, W – весовые коэффициенты нейронной сети.

Это полиномиальное уравнение из N переменных, где N – количество параметров в сети. Хотя такое уравнение можно решить для $N = 2$ или $N = 3$, это трудновыполнимо для реальных нейронных сетей, где число параметров никогда не бывает меньше нескольких тысяч и часто может составлять

несколько десятков миллионов [16]. Вместо этого можно использовать следующий алгоритм, называемый стохастическим градиентным спуском:

1. Взять батч данных, состоящих из примеров для обучения x и соответствующих меток y .
2. Выполнить прямой ход на взятом батче и получить предсказания y_{pred} .
3. Вычислить значение потерь сети на данном батче между y_{pred} и y .
4. Вычислить градиент значения потерь относительно параметров сети (обратный ход).
5. Немного изменить параметры в направлении, противоположном направлению градиента.

Процесс классификации с помощью нейронной сети делится на 2 этапа: обучение и использование. Вначале на вход нейронной сети подаются примеры для обучения. Далее сеть обучается, изменяя и настраивая весовые коэффициенты, и уже с готовыми преобразованными весами используется как классификатор.

Основными "строительными блоками" нейронных сетей являются следующие сущности:

- слой – объединенные в одну группу нейроны;
- функция активации – функция, определяющая выходной сигнал нейрона, основываясь на входном сигнале или наборе входных сигналов. например, softmax, sigmoid, relu;
- функция потерь (целевая функция) – функция, значение которой необходимо минимизировать при обучении нейронной сети;
- оптимизатор – алгоритм, определяющий то, как нейронная сеть будет обновлять свои веса, основываясь на функции потерь.

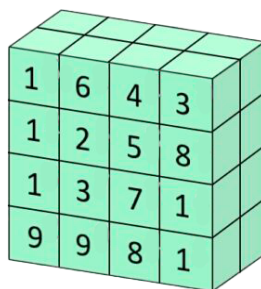
Число нейронов, количество слоев, используемые функции активации и оптимизатор называются гиперпараметрами сети.

При обучении стараются подобрать гиперпараметры так, чтобы достичь максимальной производительности (высокой точности, минимального значения потерь и т.д.).

Одним из ключевых понятий при работе с нейронными сетями является понятие тензоров. Тензоры – это обобщение матриц на произвольное число измерений. Матрица – это двумерный тензор.

Основные типы тензоров в нейронных сетях:

- скаляры (0D тензоры). Тензор, содержащий только одно число, называется скаляром;
- векторы (1D тензоры). Массив чисел называется вектором, или одномерным тензором. Считается, что одномерный тензор имеет ровно одну ось;
- матрицы (2D тензоры). Массив векторов называется матрицей, или двумерным тензором. Матрица имеет две оси (часто называемые строками и столбцами);
- 3D тензоры. Если поместить матрицы в новый массив, то получится 3D тензор, который можно визуально интерпретировать как куб чисел. Помещая трехмерные тензоры в массив, можно создать четырехмерный тензор, и так далее. Ниже на рисунке 3 приведена визуальная интерпретация 3D тензора.



A 3D visualization of a 4x4x2 tensor, represented as a cube of light green cells. The cube is composed of two 4x4 matrices stacked on top of each other. The numbers in the cells are as follows:

Row	Col 1	Col 2	Col 3	Col 4
1	1	6	4	3
2	1	2	5	8
3	1	3	7	1
4	9	9	8	1

Рисунок 3 – Визуальная интерпретация 3D тензора

Тензор определяется тремя ключевыми атрибутами:

- количество осей (ранг). Например, трехмерный тензор имеет три оси, а матрица – две оси;

- форма – это кортеж целых чисел, который описывает, сколько измерений имеет тензор по каждой оси. Например, матрица с 3 строками и 5 столбцами имеет форму (3,5);
- тип данных. Тип данных, содержащихся в тензоре; например, тип тензора может быть float32, uint8, float64 и так далее. В редких случаях можно встретить тензор char.

Все операции, производимые нейронной сетью, она производит над тензорами. При работе с текстовыми данными используются 2D тензоры.

4.3 Предобработка текстового набора данных

В настоящее время для обработки текстов на естественном языке всё чаще применяются нейронные сети. Однако перед подачей текста на их вход его необходимо преобразовать в двумерные числовые тензоры, поскольку нейронные сети не могут работать с необработанным текстом. Кроме того, чтобы получить более качественные результаты, текст, являющийся для них входными данными необходимо предобработать.

Различные методы предобработки можно поделить на следующие группы:

- простые методы предобработки;
- методы нормализации;
- методы векторизации;
- методы балансирования.

Рассмотрим каждую группу методов подробнее.

4.3.1 Простые методы предобработки

К простым методам предобработки относятся следующие методы:

- удаление дубликатов;
- удаление стоп-слов;
- удаление символов пунктуации;
- удаление чисел;
- удаление излишних пробельных символов;
- приведение слов к единому буквенному регистру.

Рассмотрим каждый из этих методов подробнее.

Удаление дубликатов

Набор данных может содержать в себе дубликаты примеров. Удаление дубликатов направлено на нахождение таких примеров и их последующее удаление.

Удаление стоп-слов

Стоп-слова – это слова, которые часто встречаются в естественном языке, но при этом не несут большой смысловой нагрузки. Удаление стоп-слов направлено на нахождение таких слов в тексте и их последующее удаление.

К стоп-словам в русском языке можно отнести предлоги, суффиксы, причастия, междометия, частицы и т.д., например, «а», «но», «и».

Стоит отметить, что не существует ни единого универсального списка стоп-слов, используемого всеми инструментами обработки естественного языка, ни правил для их идентификации. Поэтому в качестве стоп-слов для конкретной задачи может быть выбрана любая группа слов.

Удаление символов пунктуации

Символы пунктуации - это символы письменности, служащие для более точной передачи на письмо разговорной речи. Они могут обозначать интонацию, указывать связь между словами, выделять части текста.

В русском языке к символам пунктуации относятся: «.», «?», «!», «:», «;», «,», «-», «—», «(», «)».

Чаще всего смысл текста остается понятным и после удаления из него символов пунктуации. Удаление символов пунктуации направлено на нахождение таких символов в тексте и их последующее удаление.

Удаление чисел

В зависимости от конкретной задачи числа в тексте могут не нести в себе особо значимой информации, поэтому их можно удалить. Удаление чисел направлено на нахождение чисел в тексте и их последующее удаление.

Удаление излишних пробельных символов

Пробельный символ - это любой символ, который представляет собой горизонтальное или вертикальное пространство в типографике. При отображении пробельный символ не соответствует какому-либо видимому знаку, но обычно занимает определенную область на странице. Например, пробел, горизонтальная табуляция, вертикальная табуляция, возврат каретки, подача строки.

Как правило, слова в тексте разделяются с помощью единственного пробела. Кроме того, текст может содержать в себе другие пробельные символы, которые по сути не несут в себе значимой информации и могут быть заменены на одиночный пробел.

Ведущие и конечные пробелы в тексте так же в большинстве случаев не играют роли, поэтому их можно удалить.

Удаление излишних пробельных символов направлено на следующее:

- нахождение последовательности из двух и более пробелов и их замену на одиночный пробел;
- замену пробельных символов, отличных от пробела на пробел;
- удаление ведущих и конечных пробелов.

Приведение слов к единому буквенному регистру

Буквенный регистр означает размер и форму буквы, используемой в письме. Обычно различают верхний и нижний регистры. Например, предложение «На улице светит солнце» в нижнем регистре будет иметь вид «на улице светит солнце», а в верхнем – «НА УЛИЦЕ СВЕТИТ СОЛНЦЕ».

Цель такого разграничения - облегчить чтение, поскольку эти два регистра позволяют обозначать различные идеи.

Выбор буквенного регистра чаще всего определяется грамматикой языка. В русском языке, например, заглавная буква обычно используется в начале каждого предложения и для обозначения имен собственных.

Однако при составлении словаря для нейронной сети одни и те же слова в различном регистре являются разными словами. Это способствует увеличению

размера словаря и снижению обобщающей способности нейронной сети. Чтобы избежать этого буквы в словах приводят к единому регистру.

В основном выбор единого регистра не имеет значения, но поскольку в естественном языке большинство букв в предложении пишется в нижнем регистре, то с точки зрения вычислительных затрат эффективнее приводить слова именно к нему.

4.3.2 Методы нормализации

Нормализация текста - это процесс приведения слов в тексте к единой канонической форме. Он часто является важным этапом подготовки текстового набора данных. Нормализация текста сводит различные формы слова к единой форме, если они означают одно и то же, что в свою очередь упрощает обучение модели и может повысить ее производительность.

Однако прежде чем приступить к нормализации, текст необходимо токенизировать.

Токенизация — это процесс разбиения текстового документа на отдельные слова, которые называются токенами.

Например, дана строка «На улице светит солнце» при ее токенизации будут получены следующие токены: «На», «улице», «светит», «солнце».

Существует 2 основных метода нормализации текста – стемминг и лемматизация. Рассмотрим их подробнее.

Стемминг

Стемминг – это процесс нахождения основы слова для заданного исходного слова. Основа слова не всегда совпадает с его корнем. Это неизменяемая при склонении часть. Сам термин стемминг (stemming) образован от слова «stem» – ствол, стебель, основа.

Например, при стемминге слов «виноградная», «бежал» и «красиво» они примут вид «виноградн», «бежа», «красив».

Русский язык относится к группе флективных синтетических языков, то есть языков, в которых преобладает словообразование с использованием аффиксов, сочетающих сразу несколько грамматических значений (например, в

слове «добрый» — окончание «ый» указывает одновременно на единственное число, мужской род и именительный падеж), поэтому данный язык допускает использование алгоритмов стемминга.

Наиболее распространенными стеммерами для русского языка являются Стеммер Портера и Snowball.

Стеммер Портера - это алгоритм стемминга, опубликованный Мартином Портером в 1980 году.

Основная идея стеммера Портера заключается в том, что существует ограниченное количество словообразующих суффиксов, и стемминг слова происходит без использования каких-либо словарей основ: только множество существующих суффиксов и вручную заданные правила.

Алгоритм состоит из пяти шагов. На каждом шаге отсекается словообразующий суффикс, и оставшаяся часть проверяется на соответствие правилам (например, для русских слов основа должна содержать не менее одной гласной). Если полученное слово удовлетворяет правилам, происходит переход на следующий шаг. Если нет — алгоритм выбирает другой суффикс для отсечения. На первом шаге отсекается максимальный формообразующий суффикс, на втором — буква «и», на третьем — словообразующий суффикс, на четвёртом — суффиксы превосходных форм, «ь» и одна из двух «н»[13].

Данный алгоритм часто обрезает слово больше необходимого, что затрудняет получение правильной основы слова, например кровать->крова (при этом реально неизменяемая часть — кроват, но стеммер выбирает для удаления наиболее длинную морфему). Также стеммер Портера не справляется со всевозможными изменениями корня слова (например, выпадающие и беглые гласные).

Стеммер Snowball ("Снежок") является улучшенной версией стеммера Портера, поэтому он также известен как Porter2. Разница в точности определения основы слова между ними составляет около 5%.

Лемматизация

Лемматизация – это процесс приведения словоформы к лемме – её словарной форме. В русском языке словарными формами считаются следующие морфологические формы:

- для существительных – именительный падеж, единственное число;
- для прилагательных – именительный падеж, единственное число, мужской род;
- для глаголов, причастий, деепричастий – глагол в неопределённой форме несовершенного вида.

Например, при лемматизации слово «работы» примет вид «работа».

Алгоритм лемматизации основан на поиске наиболее подходящего варианта слова по словарю.

В отличие от стемминга, лемматизация зависит от правильной идентификации предполагаемой части речи и значения слова в предложении, а также в более широком контексте, окружающем это предложение, например, в соседних предложениях или даже во всем документе. Таким образом, стеммер работает с одним словом без знания контекста и поэтому не может различать слова, которые имеют разные значения в зависимости от части речи.

Стемминг производится быстрее, чем лемматизация, однако, лемматизация дает более качественные результаты.

4.3.3 Методы векторизации

Нейронные сети работают с числовыми тензорами – они не могут работать с необработанным текстом. Поэтому прежде чем подать на вход нейронной сети текст, его необходимо преобразовать в числовой тензор.

Процесс преобразования текста в числовой тензор называется векторизацией.

При работе с текстовыми данными используются 2D тензоры.

Основными методами векторизации являются:

- мешок слов;
- TF-IDF;

– word2vec.

Рассмотрим каждый из них подробнее.

Мешок слов

Одним из простейших методов векторизации текста является представление текста в виде мешка слов (BoW). Вектор BoW имеет длину всего словаря, т.е. набора уникальных слов в корпусе.

При векторизации данным методом позиция числа в векторе означает определенное слово, а само число – сколько раз слово встречается в предложении.

Например, дано предложение «на улице светит солнце, но на улице мокрые тропинки». Слову «на» соответствует 1 число в векторе, «улице» - 2, «светит» - 3, «солнце» - 4, «но» - 5, «мокрые» - 6, «тропинки» - 7. Тогда вектор будет иметь вид (2, 2, 1, 1, 1, 1, 1).

Достоинства:

- простота и понятность. Мешок слов - это простое представление текстовых данных, которое легко понять и реализовать.

Недостатки:

- нечувствительность к порядку слов. Мешок слов рассматривает все вхождения слова как эквивалентные, независимо от порядка их появления в предложении. Это означает, что он не может отразить отношения между словами в предложении и смысл, который они передают;
- разреженность векторов. Разреженный вектор – это вектор, содержащий преимущественно нули. Если корпус содержит большое количество уникальных слов, векторы, полученные с помощью мешка слов, будут иметь высокую разреженность. Разреженные векторы требуют больше памяти и вычислительных ресурсов при обучении нейронной сети.

TF-IDF

Одним из популярных методов векторизации текста является метод векторизации TF-IDF. В контексте TF-IDF под документом понимается текст.

TF-IDF (от англ. TF — term frequency, IDF — inverse document frequency) — статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов. Вес некоторого слова пропорционален частоте употребления этого слова в документе и обратно пропорционален частоте употребления слова во всех документах коллекции.

Мера TF-IDF часто используется в задачах анализа текстов и информационного поиска, например, как один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации.

Предположим, нейронной сети необходимо понять смысл предложения «Сегодня и правда замечательный день». В предложении говорится о сегодняшнем дне, о том, что он замечательный. Оно имеет позитивную окраску.

При использовании метода векторизации «мешок слов» было бы лишь подсчитано количество вхождений каждого слова в предложении без учета их релевантности относительно всего текста - «сегодня» было бы так же важно, как и «и». Кроме того, повторяющимся словам была бы присвоена большая важность, по сравнению с менее повторяющимся, но действительно важными, такими, как «замечательный», «дождь», имена и т.д.

Метод векторизации TF-IDF позволяет обойти ограничения «мешка слов», вводя понятие «inverse document frequency» (обратная частота документа).

TF-IDF рассчитывается следующим образом (формулы 2, 3, 4):

$$TF = \frac{n_t}{\sum_k n_k}, \quad (2)$$

где t — это определенное слово, n_t — это число вхождений слова t в документ, k — количество документов, n_k — количество слов в документе k .

$$IDF = \log \frac{k}{n_t}, \quad (3)$$

где k – количество документов, t – это определенное слово, n_t – это число вхождений слова t в документ.

$$TF - IDF = TF * IDF, \quad (4)$$

где TF – это частота слова, IDF – это обратная частота документа.

Другими словами, TF позволяет ответить на вопрос «как часто используется слово в документе?», а IDF - «как часто встречается слово среди всех документов?»

Рассмотрим пример вычисления $TF-IDF$.

Дано 3 предложения:

1. Сегодня будет дождь.
2. Сегодня я не пойду гулять.
3. Я буду смотреть сериал.

Выделим токены (таблица 2).

Таблица 2 – Выделение токенов

Токен	Встречаемость в 1 предложении	Встречаемость в 2 предложении	Встречаемость в 3 предложении
Сегодня	1	1	0
Будет	1	0	0
Дождь	1	0	0
Я	0	1	1
Не	0	1	0
Пойду	0	1	0
Буду	0	0	1
Смотреть	0	0	1
Сериал	0	0	1

Найдем TF (таблица 3)

Таблица 3 – Нахождение частоты слова

Токен	TF в 1 предложении	TF в 2 предложении	TF в 3 предложении
Сегодня	0,33	0,2	0
Будет	0,33	0	0
Дождь	0,33	0	0
Я	0	0,2	0,25
Не	0	0,2	0
Пойду	0	0,2	0
Буду	0	0	0,25
Смотреть	0	0	0,25
Сериал	0	0	0,25

Найдем IDF (таблица 4)

Таблица 4 – Нахождение обратной частоты документа

Токен	IDF
Сегодня	0,18
Будет	0,48
Дождь	0,48
Я	0,18
Не	0,48
Пойду	0,48
Буду	0,48
Смотреть	0,48
Сериал	0,48

Найдем TF-IDF (таблица 5)

Таблица 5 – Нахождение TF-IDF

Токен	TF-IDF в 1 предложении	TF-IDF в 2 предложении	TF-IDF в 3 предложении
Сегодня	0,0594	0,036	0
Будет	0,1584	0	0
Дождь	0,1584	0	0
Я	0	0,036	0,045
Не	0	0,096	0
Пойду	0	0,096	0
Буду	0	0	0,12
Смотреть	0	0	0,12
Сериал	0	0	0,12

Достоинства:

- учитывает не только конкретный документ, в котором встречается слово, но и другие документы коллекции.

Недостатки:

- частота встречаемости слова далеко не самый надёжный показатель релевантности, особенно для русского языка. Можно составить документ, в котором релевантное слово не будет повторяться (с использованием синонимов), или же, наоборот, текст будет перегружен омонимами нерелевантного слова;
- совершенно бесполезный текст, буквально перегруженный ключевыми словами, позволит обхитрить данную оценку.

Word2Vec

Embedding — это векторное представление слова.

Word2Vec — это группа родственных моделей, которые используются для создания embedding'ов. Эти модели представляют собой двухслойные нейронные сети, которые обучены восстанавливать лингвистические контексты слов. Word2Vec принимает на вход набор текстов и создает векторное

пространство, обычно состоящее из нескольких сотен измерений, причем каждому уникальному слову присваивается соответствующий вектор в этом пространстве.

Существуют 2 основные архитектуры Word2Vec: CBOW и Skip-gram. В обеих архитектурах Word2Vec рассматривает как отдельные слова, так и скользящее окно контекстных слов, окружающих эти отдельные слова. В модели Skip-gram по слову предсказываются слова из его контекста, а в модели CBOW по контексту подбирается наиболее вероятное слово. CBOW работает быстрее, в то время как skip-gram лучше справляется с нечастыми словами.

После обучения модели, выученные embedding'и слов располагаются в векторном пространстве таким образом, что слова, имеющие общие контексты в наборе текстов, то есть семантически и синтаксически схожие слова, располагаются близко друг к другу, и, наоборот, несхожие слова располагаются дальше друг от друга.

Достоинства:

- способен улавливать отношения между различными словами, включая их синтаксические и семантические взаимосвязи;
- векторы Embedding обладают малым размером, в отличие от большинства других алгоритмов, где размер вектора пропорционален размеру словаря.

Недостатки:

- для высокой производительности требует достаточно большой набор текстов.

4.3.4 Методы балансирования

При решении задачи классификации необходимо учитывать проблему несбалансированности классов.

Несбалансированность классов – это проблема, при которой на один пример в одном классе приходится десятки, сотни или даже тысячи примеров в другом.

Использование наборов данных с несбалансированными классами представляет собой проблему для моделирования, т.к. большинство алгоритмов машинного обучения спроектированы вокруг предположения о равном количестве экземпляров для каждого класса [22]. Это ведет к тому, что модель имеет крайне слабую обобщающую способность, особенно для класса, находящегося в меньшинстве.

Если использовать для обучения нейронной сети данные, несбалансированные по классам, например, набор данных в котором 60000 примеров относятся к одному классу и лишь 100 примеров – к другому, то нейронная сеть будет в большинстве случаев предсказывать лишь 1-ый класс, в то время как 2-ой будет предсказываться гораздо реже.

Для решения проблемы несбалансированности классов можно использовать следующие методы:

- даунсемплинг;
- апсемплинг;
- приведение количества примеров в каждом классе к среднему значению.

Рассмотрим каждый из методов подробнее.

Даунсемплинг

Даунсемплинг – это метод балансирования набора данных, при котором количество примеров в каждом классе уменьшается до количества примеров в классе с наименьшим количеством примеров.

Например, пусть дан набор данных (таблица 6).

Таблица 6 – Несбалансированный набор данных

Текст	Класс
Задача о поставках	МА
Адаптивные технологии в современных автоматизированных обучающих системах	КИИН

Продолжение таблицы 6

Интеллектуализация пользовательских интерфейсов информационных систем	КИИН
Синтез полиариленфталидов перспективных в качестве “умных” полимеров	ХН
Оценка термодинамических параметров реакций синтеза акрилонитрила методами квантовой химии	ХН
Термоокисление в-каротина в растворе	ХН

В классе МА содержится 1 пример, в КИИН – 2 примера, а в ХН – 3 примера. Класс МА является классом с наименьшим количеством примеров. Следовательно, после даунсемплинга набор данных примет следующий вид (таблица 7).

Таблица 7 – Набор данных после даунсемплинга

Текст	Класс
Задача о поставках	МА
Адаптивные технологии в современных автоматизированных обучающих системах	КИИН
Синтез полиариленфталидов перспективных в качестве “умных” полимеров	ХН

Апсемплинг

Апсемплинг – это метод балансирования набора данных, при котором количество примеров в каждом классе увеличивается до количества примеров в классе с наибольшим количеством примеров.

При апсемплинге дублируемые в каждом классе примеры выбираются случайным образом.

Например, пусть дан набор данных (см. таблицу 6).

Класс ХН является классом с наибольшим количеством примеров. Следовательно, после апсемплинга набор данных примет следующий вид (таблица 8).

Таблица 8 – Набор данных после апсемплинга

Текст	Класс
Задача о поставках	МА
Задача о поставках	МА
Задача о поставках	МА
Адаптивные технологии в современных автоматизированных обучающих системах	КИИН
Интеллектуализация пользовательских интерфейсов информационных систем	КИИН
Адаптивные технологии в современных автоматизированных обучающих системах	КИИН
Синтез полиарилефталидов перспективных в качестве “умных” полимеров	ХН
Оценка термодинамических параметров реакций синтеза акрилонитрила методами квантовой химии	ХН
Термоокисление в-каротина в растворе	ХН

Приведение количества примеров в каждом классе к среднему значению

Данный метод является комбинацией даунсемплинга и апсемплинга.

Сначала высчитывается среднее количество примеров в каждом классе.

Затем для классов, в которых количество примеров меньше среднего производится апсемплинг, а для классов, в которых количество примеров больше среднего производится даунсемплинг.

4.4 Виды нейронных сетей для классификации текста

Классификацию текста можно производить с помощью различных видов нейронных сетей, однако, наиболее распространенными являются рекуррентные нейронный сети и нейронные сети с долгой краткосрочной памятью. Рассмотрим эти виды подробнее.

4.4.1 Рекуррентные нейронные сети

Рекуррентная нейронная сеть (РНС) - это вид нейронной сети, в которой выход предыдущего шага подается на вход текущего шага. В традиционных нейронных сетях все входы и выходы независимы друг от друга, но в случаях, когда требуется предсказать следующее слово предложения, требуются предыдущие слова, и, следовательно, есть необходимость помнить предыдущие слова. Таким образом, появилась RNN, которая решила эту проблему с помощью скрытого слоя. Основной и наиболее важной особенностью RNN является ее скрытое состояние, которое запоминает некоторую информацию о последовательности. Это состояние также называют состоянием памяти, поскольку оно запоминает предыдущий вход в сеть. Сеть использует одни и те же параметры для каждого входа, поскольку она выполняет одну и ту же задачу на всех входах или скрытых слоях для получения выхода. Это уменьшает сложность параметров, в отличие от других нейронных сетей.

Однако РНС, как правило, сталкиваются с двумя проблемами, известными как взрывающиеся градиенты и исчезающие градиенты. Эти проблемы определяются размером градиента, который представляет собой наклон функции потерь вдоль кривой ошибок. Когда градиент слишком мал, он продолжает уменьшаться, обновляя весовые параметры, пока они не станут незначительными - т.е. 0. Когда это происходит, алгоритм больше не обучается. Взрывные градиенты возникают, когда градиент слишком велик, создавая неустойчивую модель. В этом случае весовые параметры модели становятся слишком большими и в конечном итоге представляются как NaN.

4.4.2 Нейронные сети с долгой краткосрочной памятью

Сети с долгой краткосрочной памятью (LSTM) - это расширение для RNN, которое, по сути, расширяет память. Поэтому они хорошо подходят для обучения на основе важных событий, между которыми существует очень большой промежуток времени.

Модули LSTM используются в качестве строительных единиц для слоев RNN, часто называемых LSTM-сетью.

LSTM позволяют RNN запоминать входные данные в течение длительного периода времени. Это происходит потому, что LSTM содержат информацию в памяти, подобно памяти компьютера. LSTM может читать, записывать и удалять информацию из своей памяти.

Эту память можно представить, как ячейку с воротами, причем "воротами" означает, что ячейка решает, хранить или удалять информацию (т.е. открывает ворота или нет), основываясь на важности, которую она придает информации. Присвоение важности происходит с помощью весов, которым также обучается алгоритм. Это означает, что со временем он узнает, какая информация важна, а какая нет.

В ячейке долговременной памяти есть трое ворот: входные ворота, ворота забывания и выходные ворота. Эти ворота определяют, следует ли впустить новую информацию (входные ворота), удалить ее, потому что она не важна (ворота забывания), или позволить ей повлиять на выход в текущий момент времени (выходные ворота).

Таким образом, LSTM решают проблему исчезающих градиентов.

4.5 Решение задачи классификации

В машинном обучении многоклассовая классификация - это проблема отнесения экземпляров к одному из трех или более классов (отнесение экземпляров к одному из двух классов называется бинарной классификацией).

Многоклассовая классификация предполагает, что каждому экземпляру присваивается один и только один класс - экземпляр не может относиться к двум классам одновременно.

Как правило, для решения задачи многоклассовой классификации с помощью нейронных сетей последним слоем используют слой с функцией активации softmax.

Функция активации softmax присваивает вероятность принадлежности экземпляра к каждому из рассматриваемых классов. Все вероятности в сумме должны давать 1.0. Количество нейронов в слое с softmax равно количеству рассматриваемых классов.

4.6 Оценка качества обучения

При обучении набор данных разбивается на 3 части: тренировочную, тестовую и валидационную выборки.

Обучение модели происходит на тренировочной выборке, ее оценка во время обучения – на валидационной. После того, как модель обучена, её оценивают на тестовой выборке. Такое разделение связано с тем, что при обучении архитектор, основываясь на производительности модели на валидационной выборке, пытается изменить ее так, чтобы улучшить ее производительность. При этом возможен такой феномен, как информационная утечка. Каждый раз при изменении конфигурации в модель будет "просачиваться" информация о валидационной выборке, что может привести к тому, что модель будет показывать хорошие результаты на валидационной выборке, но плохие – на новых данных [10].

Процесс обучения модели можно рассматривать как задачу минимизации функции потерь (формула 5)

$$E(t) = \frac{1}{P} \sum_{i=1}^P (d_i(t) - y_i(t))^2, \quad (5)$$

где t — номер итерации обучения, P — количество отработанных моделью примеров, d_i — желаемый выход модели, y_i — реальный выход модели. В процессе обучения значение потерь постепенно уменьшается. В то же время после определенной итерации может наблюдаться возрастание значения потерь на валидационной выборке, что говорит об ухудшении обобщающей способности модели.

Такой феномен называется переобучением и случается тогда, когда модель уже не извлекает какие-либо правила из подаваемых ей на вход данных, а запоминает то, какой результат соответствует определенному примеру. В большинстве случаев переобучение возникает из-за того, что модель имеет слишком большое число параметров [10]. Ниже на рисунке 4 приведено проявление феномена переобучения на графике значений потерь.

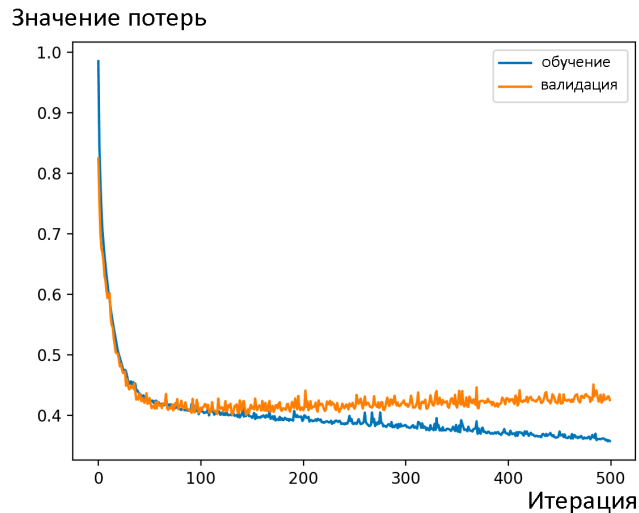


Рисунок 4 – Проявление феномена переобучения на графике значений потерь

Для контролирования переобучения можно использовать различные техники, такие как принудительная остановка обучения после определенной эпохи, регуляризация и улучшение данных [33].

Так как в данной работе используется многоклассовая классификация, то для нее в качестве функции потерь была выбрана категориальная кросс-энтропия (формула 6).

$$LOSS = - \sum_i^n y_i \log(\hat{y}_i) \quad (6)$$

где LOSS – значение потерь, n – количество примеров, y_i – вероятность i -го примера, \hat{y}_i – предсказанная вероятность i -го примера.

Кроме значения потерь при оценке качества обучения используются различные количественные меры, называемые метриками.

Метрики

Одними из наиболее распространенных при оценке качества модели для решения задачи классификации являются следующие метрики:

- точность (accuracy);
- чувствительность (sensitivity);
- специфичность (specificity);

Рассмотрим каждую метрику.

Точность

Точность определяется как соотношение числа правильных предсказаний ко всему числу предсказаний (формула 7)

$$ACC = \frac{TP + TN}{P + N}, \quad (7)$$

где ACC – точность, TP – количество истинно положительных предсказаний, TN – количество истинно отрицательных предсказаний, P – количество положительных предсказаний, N – количество отрицательных предсказаний.

Чувствительность

Чувствительность определяется как отношение истинно положительных результатов к сумме истинно положительных и ложно отрицательных результатов (формула 8)

$$TPR = \frac{TP}{TP + FN}, \quad (8)$$

где TPR – чувствительность, TP – количество истинно положительных предсказаний, FN – количество ложно отрицательных предсказаний.

Специфичность

Специфичность определяется как отношение истинно отрицательных результатов к сумме истинно отрицательных и ложно положительных результатов (формула 9)

$$TNR = \frac{TN}{TN + FP}, \quad (9)$$

где TNR – специфичность, TN – количество истинно отрицательных предсказаний, FP – количество ложно положительных предсказаний.

Помимо метрик при оценке качества обучения модели для задачи классификации используют матрицу ошибок.

5 ПРАКТИЧЕСКАЯ ЧАСТЬ

5.1 Проектирование функциональности программного обеспечения

Исходя из предъявляемых к программному обеспечению требований, описанных в разделе 1, можно выделить следующую функциональность:

- предобработка текста;
- обучение модели;
- оценка качества модели;
- классификация текстовых данных.

Ниже на рисунке 5 приведено представление вариантов использования.

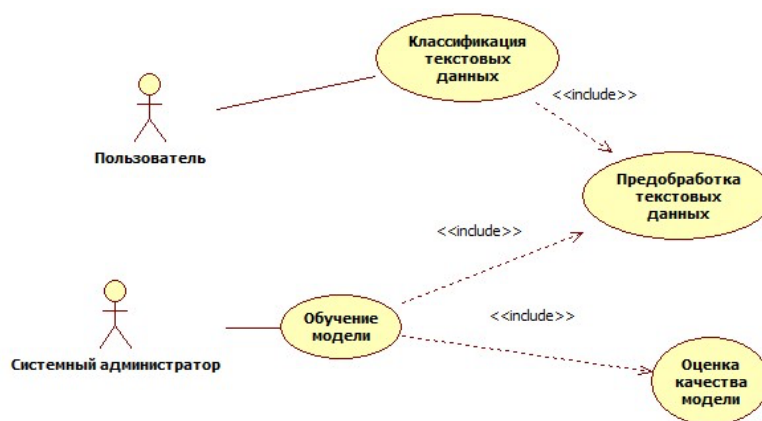


Рисунок 5 – Представление вариантов использования

На рисунке 5 представлено 2 актера «Пользователь» и «Системный администратор». Актер «Пользователь» инициирует 1 прецедент – «Классификация текстовых данных». Прецедент «Классификация текстовых данных» включает в себя прецедент «Предобработка текстовых данных», что показано отношением включения. Актер «Системный администратор» инициирует 1 прецедент – «Обучение модели». Прецедент «Обучение модели» включает в себя прецеденты «Предобработка текстовых данных» и «Оценка качества модели», что показано отношением включения. Ниже в таблицах 9-10 приведена спецификация указанных прецедентов.

Таблица 9 – Спецификация варианта использования «Обучение модели»

Элемент	Описание
Название	Обучение модели
Действующие лица	Системный администратор
Краткое описание	Системный администратор производит обучение модели
Предусловия	Системный администратор использует компьютер для обучения моделей
Постусловия	Модель обучена
Нормальное течение	1. Системный администратор запустил ноутбук в Google Colab для обучения моделей. 2. Системный администратор получил обученную модель.
Альтернативные течения	1. Системный администратор запустил приложение для обучения моделей, используя неверные входные данные. (а) Выдается сообщение об ошибке. 2. При сохранении выходных данных произошли ошибки. (а) Выдается сообщение об ошибке.
Приоритет (Критично Важно Желательно):	Критично
Частота использования (Всегда Часто Иногда Редко Один раз):	Редко

Таблица 10 – Спецификация варианта использования «Классификация текстовых данных»

Элемент	Описание
Название	Классификация текстовых данных
Действующие лица	Пользователь
Краткое описание	Пользователь производит классификацию текстовых данных
Предусловия	1. Веб-приложение запущено. 2. Открыта страница для классификации текстовых данных.
Постусловия	Модель вывела предполагаемый класс и точность предположения
Нормальное течение	1. Пользователь загрузил текстовые данные для классификации. 2. Пользователь получил предполагаемый класс и точность предположения.

Продолжение таблицы 10

Альтернативные течения	1. Пользователь загрузил некорректные данные. (а) Выдается сообщение об ошибке. 2. Во время загрузки данных была потеряна связь с сервером веб-приложения. (а) Выдается сообщение об ошибке.
Приоритет (Критично Важно Желательно):	Критично
Частота использования (Всегда Часто Иногда Редко Один раз):	Часто

5.2 Разработка архитектуры программного обеспечения

Одним из требований, предъявляемых к разрабатываемому программному обеспечению является возможность доступа к нему любого пользователя, которому необходимо произвести классификацию текстовых данных.

С другой стороны, программное обеспечение должно позволять обучать модель для классификации. Как было сказано в разделе 3, обучение модели является крайне затратным процессом с точки зрения производимых вычислений, к тому же для этого требуются огромные объемы данных, занимающие большое количество места на жестком диске.

В связи с этим было принято решение разработать программное обеспечение, состоящее из трех частей: модуль предобработки текстовых данных, приложение для обучения моделей и веб-приложение для классификации текстовых данных.

Использование веб-приложения обладает следующими преимуществами [31]:

- отсутствие необходимости в дорогостоящем аппаратном обеспечении.

Так как все вычисления производятся на сервере, то пользователю нет нужды покупать мощный компьютер;

- актуальность модели. Пользователи всегда используют самую новую модель;
- отсутствие необходимости в обновлении программного обеспечения на стороне пользователя. Достаточно обновить веб-приложение на сервере и все изменения сразу станут доступны всем его пользователям.

Приложение для обучения моделей будет представлено в виде ноутбука Google Colab и будет размещено в Google Drive. Среда Google Colab предоставляет видеокарты с достаточно высокой производительностью, чтобы обучить модель, основанную на рекуррентных нейронных сетях. Также в Google Drive будут храниться и все данные для обучения модели.

Веб-приложение будет располагаться на отдельном компьютере, выполняющем роль сервера.

Пользователям, для использования веб-приложения, будет достаточно лишь браузера.

Коммуникация между клиентскими рабочими станциями, сервером веб-приложения и средой Google Colab осуществляется с помощью протокола HTTP.

Рассмотрев архитектуру создаваемого программного обеспечения, перейдем к его непосредственной разработке.

5.3 Разработка модуля предобработки текстового набора данных

Приложение для обучения моделей и веб-приложение для классификации текстовых данных используют одни и те же методы предобработки текстового набора данных, в связи с чем было принято решение выделить эти методы в отдельный модуль.

Модуль предобработки текстовых данных разработан с использованием языка программирования Python и библиотек NLTK, PyMorphy2 и Gensim. В качестве среды разработки используется IntelliJ PyCharm 2020.1 Community Edition.

Сам модуль представляет из себя файл с расширением *.py.

5.3.1 Описание входных и выходных данных

Центральным классом в модуле, выполняющим предобработку текстового набора данных является класс Preprocessor – препроцессор текстового набора данных.

При предобработке текстовых данных для обучения на вход ему подается набор данных в виде таблицы Pandas DataFrame. Таблица должна иметь столбцы “text” и “class”, в которых указываются тексты и классы, к которым они принадлежат, соответственно.

В качестве выходных данных служит предобработанный набор данных в виде таблицы Pandas DataFrame, токенайзер, полученный в ходе предобработки, и, если применялся один из методов векторизации word2vec, то матрица весов для слоя Embedding.

При предобработке текстовых данных для классификации на вход ему подается набор данных в виде таблицы Pandas DataFrame. Таблица должна иметь столбец “text”, в котором указываются тексты для классификации.

В качестве выходных данных служит предобработанный набор данных в виде таблицы Pandas DataFrame.

5.3.2 Разработка алгоритмов

Разработаем алгоритмы, необходимые для реализации прецедента «Предобработка текстовых данных», описанного в разделе 5.1.

Модуль предобработки текстового набора данных предоставляет следующие методы предобработки.

Простые методы предобработки набора данных

- удаление дубликатов.

Простые методы предобработки текстовых данных

- удаление символов пунктуации;
- удаление чисел;
- удаление излишних пробельных символов;
- приведение слов к нижнему буквенному регистру;

- удаление стоп-слов.

Методы нормализации слов

- стемминг;
- лемматизация.

Методы векторизации текста

- мешок слов;
- TF-IDF;
- word2Vec (CBOW);
- word2Vec (Skip gram).

Методы балансирования текстового набора данных

- даунсемплинг;
- апсемплинг;
- приведение количества примеров в каждом классе к среднему значению.

Каждый вышеуказанный метод реализуется отдельным классом.

Класс `Preprocessor` выступает в качестве оркестратора методов предобработки – в него добавляются объекты классов различных методов предобработки, а он, уже в зависимости от их типа, решает, на каком этапе предобработки текстового набора данных их стоит использовать.

Ниже на рисунке 6 приведен алгоритм работы класса `Preprocessor` при предобработке текстового набора данных для обучения.

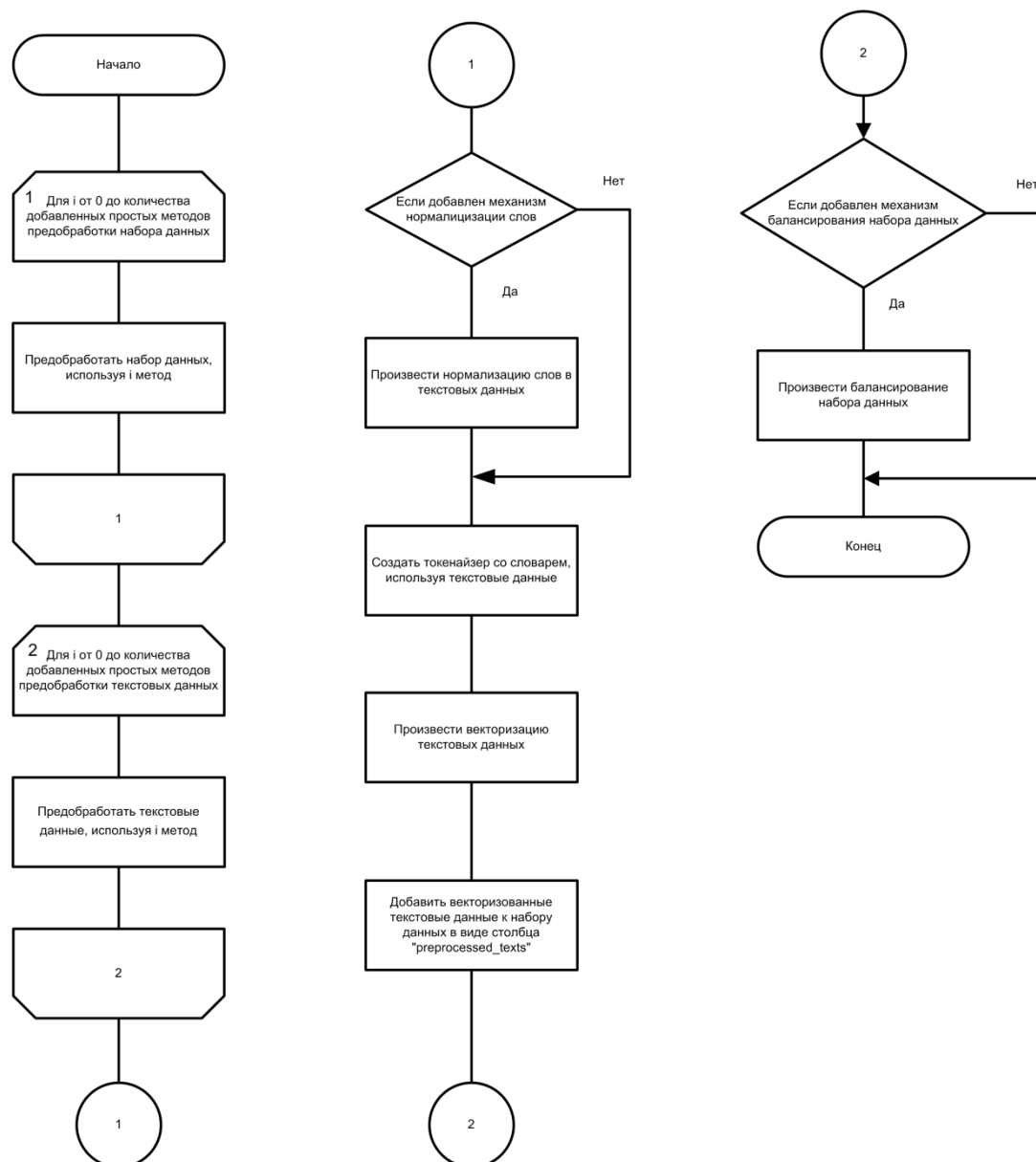


Рисунок 6 – Алгоритм работы класса Preprocessor при предобработке текстового набора данных для обучения

5.3.3 Программная реализация

Методы предобработки представлены в виде отдельных классов – «механизмов». Каждый из них имеет метод process, выполняющий соответствующую предобработку. Конкретный метод предобработки указывается через передаваемое в них значение соответствующего перечисления. Рассмотрим эти классы.

SimpleDatasetPreprocessingMechanism

Класс, реализующий простые методы предобработки набора данных.

Конструктор класса принимает на вход значение перечисления SimpleDatasetPreprocessing и, опционально, словарь с параметрами. Возможные значения SimpleDatasetPreprocessing:

- DELETE_DUPLICATES.

В словаре с параметрами через ключ “duplicate_column” можно указать название столбца, в котором необходимо искать дубликаты. По умолчанию по ключу “duplicate_column” хранится значение “text”.

Метод process обладает параметром dataset – набор данных в виде таблицы Pandas DataFrame.

На выходе ожидается предобработанный соответствующим образом набор данных в виде таблицы Pandas DataFrame.

SimpleTextPreprocessingMechanism

Класс, реализующий простые методы предобработки текстовых данных.

Конструктор класса принимает на вход значение перечисления SimpleTextPreprocessing и, опционально, словарь с параметрами. Возможные значения SimpleTextPreprocessing:

- DELETE_PUNCTUATION;
- DELETE_NUMBERS;
- DELETE_EXTRA_WHITESPACES;
- TO_LOWER_CASE;
- DELETE_STOPWORDS.

В словаре с параметрами через ключ “language” можно указать на каком языке представлены текстовые данные (необходимо для методов предобработки «Удаление стоп-слов»). По умолчанию по ключу “language” хранится значение “russian”.

Метод process обладает параметром text – строка, в которой содержится текст для предобработки.

На выходе ожидается предобработанный соответствующим образом текст в виде строки.

Для удаления чисел используется регулярное выражение «([0-9]+\.)+».

Для удаления лишних пробельных символов используется регулярное выражение «\s{2,}».

Для удаления стоп-слов используется список стоп-слов русского языка из библиотеки NLTK.

Для удаления символов пунктуации используется список символов пунктуации из библиотеки NLTK.

WordNormalizationMechanism

Класс, реализующий методы нормализации слов.

Конструктор класса принимает на вход значение перечисления WordNormalizationModeи, опционально, словарь с параметрами. Возможные значения WordNormalizationMode:

- STEMMING;
- LEMMATIZATION.

В словаре с параметрами через ключ “language” можно указать на каком языке представлены текстовые данные. По умолчанию по ключу “language” хранится значение “russian”.

Метод process обладает параметром text – строка, в которой содержится текст для предобработки.

На выходе ожидается предобработанный соответствующим образом текст в виде строки.

Для стемминга используется стеммер Snowball из библиотеки NLTK.

Для лемматизации используется библиотека PyMorphu2.

SequenceVectorizationMechanism

Класс, реализующий простые методы векторизации текста.

Конструктор класса принимает на вход значение перечисления SequenceVectorizationMode, и, опционально, словарь с параметрами. Возможные значения SequenceVectorizationMode:

- BAG_OF_WORDS;

- TF_IDF;
- WORD2VEC_CBOW;
- WORD2VEC_SKIP_GRAM.

В словаре с параметрами через ключ “num_words” можно указать размер словаря, а через ключ “max_sequence_length” – максимальный размер вектора. По умолчанию по ключам “num_words” и “max_sequence_length” хранится значение 1000.

Метод process обладает параметром texts – массив строк, которые необходимо векторизовать.

На выходе ожидается предобработанный соответствующим образом массив строк, полученный токенайзер, и, если использовался один из методов векторизации word2vec, то матрица весов для слоя Embedding.

Для методов векторизации «Мешок слов» и «TF-IDF» используется библиотека Keras.

Для методов векторизации word2vec используется библиотека Gensim.

DatasetBalancingMechanism

Класс, реализующий методы балансирования текстового набора данных.

Конструктор класса принимает на вход значение перечисления DatasetBalancingMode. Возможные значения DatasetBalancingMode:

- DOWNSAMPLING;
- UPSAMPLING;
- AVERAGING.

Метод process обладает параметрами dataset – набор данных в виде таблицы Pandas DataFrame и class_column_name – название столбца с классами.

На выходе ожидается предобработанный соответствующим образом набор данных в виде таблицы Pandas DataFrame.

Метод даунсемплинга вычисляет минимальное количество примеров в классе и для каждого класса, если это необходимо, уменьшает количество примеров до минимального.

Метод апсемплинга вычисляет максимальное количество примеров в классе и для каждого класса, если это необходимо, увеличивает количество примеров до максимального путем дублирования.

Метод приведения количества примеров в каждом классе к среднему значению вычисляет среднее значение примеров в классе и затем для каждого класса, если количество примеров в нем меньше среднего, то происходит дублирование до среднего значения, а если больше среднего, то происходит уменьшение количества примеров до среднего значения.

Preprocessor

Класс, производящий предобработку текстового набора данных. Preprocessor оркеструет механизмы предобработки в зависимости от их типа.

Метод `add_preprocessing` позволяет добавить к препроцессору механизм предобработки. Так как одновременно для нормализации слов, векторизации текста и балансирования набора данных может использоваться лишь по одному механизму, то добавление нескольких механизмов этих типов повлечет за собой их перезапись в препроцессоре – будет использоваться последний добавленный механизм. Метод принимает параметр `preprocessing_mechanism` – объект механизма предобработки.

По умолчанию, если не добавлен ни один механизм предобработки, то будет использоваться только метод векторизации текста «Мешок слов».

Метод `preprocess_dataset_for_training` производит предобработку текстового набора данных для обучения нейронной сети. В ходе его работы создается токенайзер, и, если применялся один из методов векторизации `word2vec`, матрица весов для слоя `Embedding`. Он имеет параметр `dataset` – набор данных в виде таблицы `Pandas DataFrame`, и, опционально, принимает параметры `text_column_name` – название столбца с текстами и `class_column_name` – название столбца с классами. По умолчанию `text_column_name` имеет значение “text”, а `class_column_name` – “class”. Возвращаемым значением является предобработанный набор данных в виде таблицы `Pandas DataFrame`.

Метод `preprocess_dataset_for_evaluating` производит предобработку текстового набора данных для эксплуатации нейронной сети. В ходе его работы применяется ранее созданный токенайзер. Он имеет параметр `dataset` – набор данных в виде таблицы `Pandas DataFrame`, и, опционально, принимает параметры `text_column_name` – название столбца с текстами и `class_column_name` – название столбца с классами. По умолчанию `text_column_name` имеет значение “text”, а `class_column_name` – “class”. Возвращаемым значением является предобработанный набор данных в виде таблицы `Pandas DataFrame`.

5.4 Разработка приложения для обучения моделей

Приложение для обучения моделей разработано с использованием языка программирования Python и библиотеки глубокого обучения Keras, в качестве среды разработки используется Google Colab.

Приложение представляет из себя ноутбук Jupyter Notebook (имеет расширение *.ipynb), так же запускаемый в Google Colab.

5.4.1 Описание входных и выходных данных

Входными данными приложения для обучения моделей является файл формата CSV (Comma Separated Values – файл со значениями, разделенными запятыми), соответствующий следующим критериям:

- первый столбец имеет название «text» (текст для классификации);
- столбец «text» в каждой клетке содержит текст, который необходимо классифицировать;
- второй столбец имеет название «class» (класс, к которому относится соответствующий текст);
- столбец «class» в каждой клетке содержит класс, к которому относится соответствующий текст;
- разделителями значений являются «,».

CSV-файл содержит все данные, необходимые для обучения модели. Файл должен располагаться в Google Drive по пути «/Colab Notebooks/Text

Classification/Datasets/train.csv». Ниже на рисунке 7 изображен фрагмент такого файла.

```
1 text,class
2 Безлинзовый цифровой голографический микроскоп,ЕИТН
3 Трехмерные модели трения,ЕИТН
4 Дифракция неполяризованного оптического излучения на объемных акустических волнах в кристалле ниобата лития,ЕИТН
5 Высокоинтенсивная имплантация ионов алюминия в никель и титан,ЕИТН
6 Исследование структуры течения в новой конструкции вихревой топki методом цифровой трассерной визуализации,ЕИТН
7 Устойчивость растяжения вязкопластической полосы,ЕИТН
8 Моделирование волн на поверхности цилиндрической конфигурации магнитной жидкости окружающей длинное пористое ядро,ЕИТН
9 Второй вириальный коэффициент метана,ЕИТН
10 Обобщения контактной задачи Галина и взаимодействие штампов,ЕИТН
11 Опыт разработки и применения датчиков уровня систем управления путевых железнодорожных машин,ЕИТН
12 Топологические солитоны в двухцепочечной модели ДНК,ЕИТН
13 Об устойчивости материалов к разрушению вблизи температур структурно-фазовых превращений,ЕИТН
14 Геометрия поверхности текучести и законы упрочнения в физических теориях пластичности,ЕИТН
15 Коэффициент поперечной деформации и ангармонизм колебаний решетки квазиизотропных твердых тел,ЕИТН
16 Экспериментальное исследование газового эжектора с перфорированным соплом высоконапорного потока,ЕИТН
17 Причинный анализ квантовых запутанных состояний. Ч. 2,ЕИТН
```

Рисунок 7 – Фрагмент CSV-файла

На выходе ожидаются:

- файл обученной модели с названием `trained_model.keras`;
- изображение графиков значения потерь на обучающей и валидационной выборках;
- изображение графиков точности на обучающей и валидационной выборках;
- изображение графиков чувствительности на обучающей и валидационной выборках;
- изображение графиков специфичности на обучающей и валидационных выборках;
- значение потерь, точность, чувствительность и специфичность на обучающей, валидационной и тестовой выборках.

Файл обученной модели сохраняется в Google Drive по пути «/Colab Notebooks/Text Classification/Models/trained_model.keras». В файле содержится обученная модель, использованный препроцессор, который содержит в себе необходимый токеназатор, и, если использовался метод векторизации `word2vec`, матрица весов для слоя `Embedding`.

Рассмотрев описание входных и выходных данных, перейдем к разработке алгоритмов.

5.4.2 Разработка алгоритмов

Разработаем алгоритм, необходимый для реализации прецедента «Обучение модели», описанного в 5.1.

На рисунке 8 приведен разработанный алгоритм.

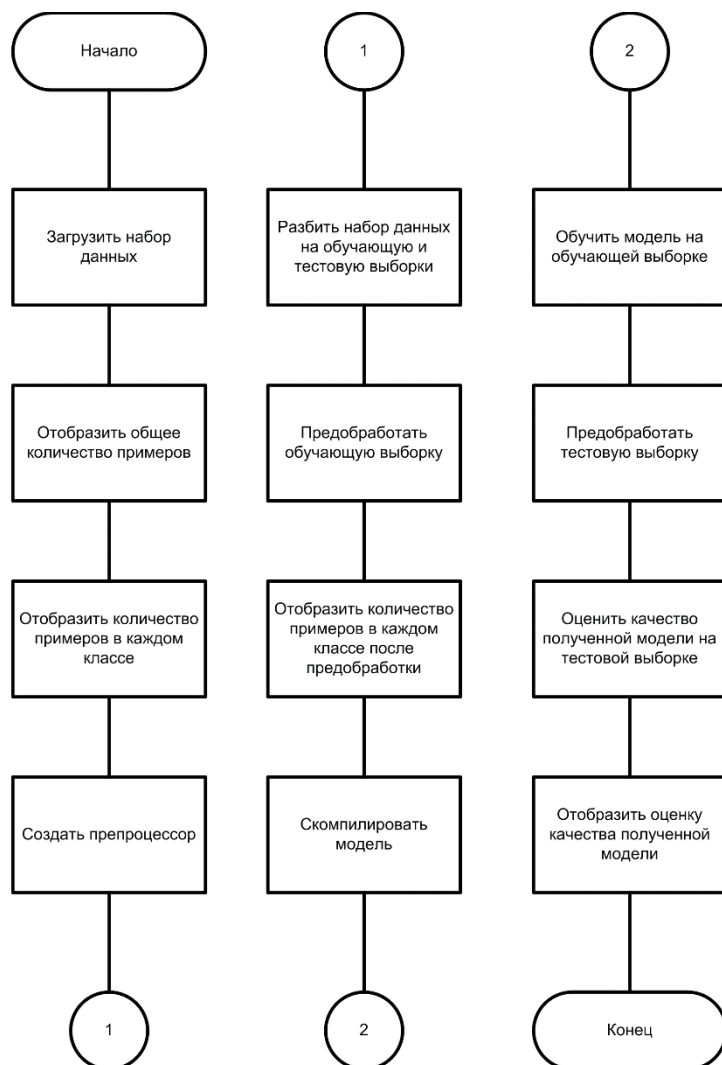


Рисунок 8 – Общий алгоритм работы приложения для обучения моделей

Перейдем к реализации разработанного алгоритма.

5.4.3 Программная реализация алгоритмов

Для реализации алгоритма, разработанного в 5.4.2, были созданы специальные методы.

Загрузка CSV-файла

Функция `load_csv(path)` позволяет загрузить содержимое CSV-файла в виде объекта класса `DataFrame` (класс, представляющий таблицу). На вход ему поступает путь к CSV-файлу, на выходе возвращается объект класса `DataFrame`.

В случае, если указанный файл загрузить не удалось, то выбрасывается исключение.

Проверка набора данных на соответствие требованиям

Функция `is_dataset_valid(dataset)` позволяет проверить соответствие загруженного набора данных требованиям, предъявленным в 5.4.1. Входной параметр: объект класса `data_frame`. Выход: `True` или `False`, в зависимости от того, прошел ли набор данных проверку.

Разбиение данных на 2 выборки

Функция `split_dataset(dataset, test_dataset_percent)` производит разбиение таблицы набора данных `data_frame` на 2 выборки – обучающую и тестовую. Входные параметры: `data_frame` – таблица с набором данных, `test_dataset_percent` – процент примеров для тестовой выборки от общего числа примеров. Выходные значения: 2 таблицы в виде `Pandas DataFrame` – для обучающей и тестовой выборок. Если `test_set_percent >= 1`, то выбрасывается исключение.

Обучение модели

Метод `train_model(data_frame, preprocessor, model, epochs, batch_size)` позволяет обучить модель. Входные параметры: `data_frame` – таблица с данными, `preprocessor` – препроцессор, `model` – модель, которую необходимо обучить, `epochs` – максимальное количество эпох, `batch_size` – размер батча. Выходные значения – обученная модель и история обучения.

Оценка качества модели

Метод `evaluate_model(test_data_frame, model)` производит оценку качества модели, применяя метрики, рассмотренные в 4.6. Отображает результаты оценки качества.

Сохранение модели

Метод `save_model(model, path)` производит сохранение обученной модели по пути `path`. Входные параметры: `model` – обученная модель, `path` – путь, по которому необходимо сохранить `model`. По умолчанию `path` имеет значение «/Colab Notebooks/Text Classification/Models/trained_model.keras». Выходное

значение отсутствует. Если при сохранении произошла ошибка, то выбрасывается исключение.

5.5 Разработка веб-приложения

Разработка веб-приложения велась на языке программирования Python с использованием фреймворков Django и Keras.

Для повышения возможности повторного использования кода, а также для разделения представления и логики веб-приложения было решено применить шаблон проектирования Model-View-Controller (MVC).

Шаблон проектирования Model-View-Controller («Модель-Представление-Контроллер») – это шаблон проектирования, позволяющий разделить программную логику на 3 взаимосвязанных элемента: модель, представление и контроллер [6].

Модель – это элемент, который напрямую управляет данными, а также содержит бизнес-логику для работы с ними.

Представление – это элемент, отвечающий за отображение данных модели пользователю.

Контроллер – это элемент, который принимает запросы от пользователя и конвертирует их в команды для модели и/или представления.

Ниже на рисунке 9 схематично представлено взаимодействие между элементами шаблона.



Рисунок 9 – Взаимодействие между элементами шаблона MVC

Такое разделение позволяет модифицировать каждый из элементов независимо и повысить возможность повторного использования кода.

5.5.1 Разработка алгоритмов

Пользователь должен иметь возможность классифицировать, как сам текст, так и тексты из CSV-файла. Исходя из спецификаций прецедентов, приведенных в 5.1, произведем разработку алгоритмов. Ниже на рисунках 10-11 приведены разработанные алгоритмы.

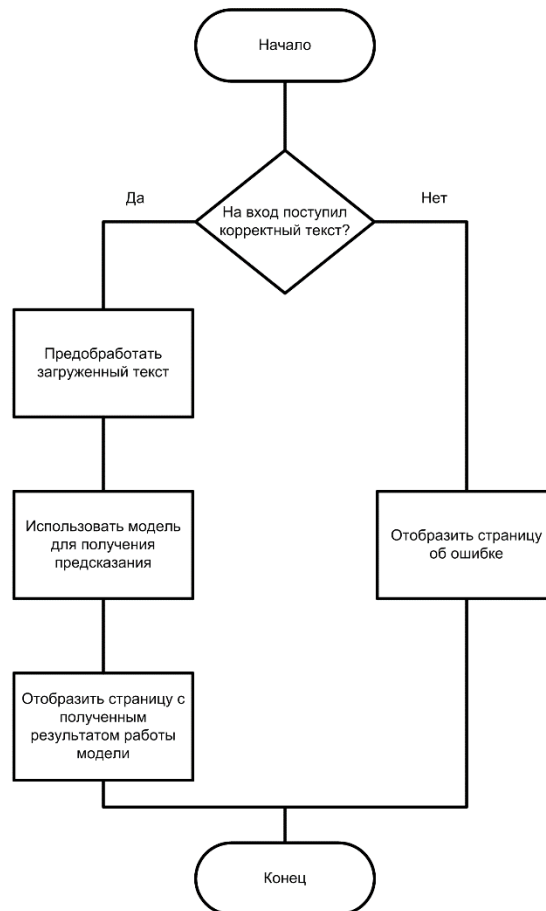


Рисунок 10 – Алгоритм классификации по поступающему на вход тексту

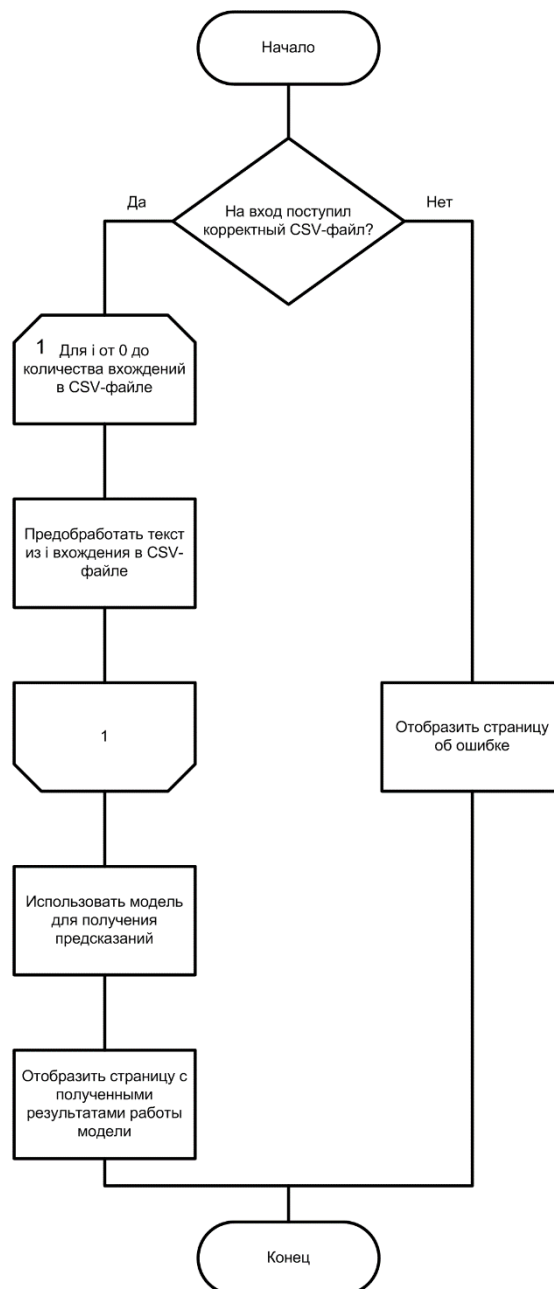


Рисунок 11 – Алгоритм классификации по поступающему на вход CSV-файлу

5.5.2 Разработка классов

Для веб-приложения были созданы классы, описание которых приведено в таблице 11.

Таблица 11 – Описание классов

Название класса	Описание
TextClassificationController	Контроллер веб-приложения, который содержит в себе логику для корректной работы с HTTP-запросами пользователей (отображение главной страницы, классификация по тексту, классификация по CSV-файлу, отображения страницы об ошибке)
CsvFileValidator	Производит проверку поданного на вход CSV-файла предъявляемым требованиям (см. раздел 5.5.3)
TextClassificationResultDto	Объект (DTO) с результатами классификации текста

5.5.3 Программная реализация

Листинги данной части программного обеспечения на языке программирования Python приведены в приложении.

При старте веб-приложения в оперативную память загружается файл с обученной моделью `trained_model.keras`.

Реализация классификации текстовых данных

В классе-контроллере `TextClassificationController` присутствуют методы `index`, `predict_by_text` и `predict_by_file`. Метод `index` обрабатывает GET-запросы, поступающие по пути <URL сервера веб-приложения>. При загрузке данного URL пользователь попадает на страницу, где ему предлагается загрузить текст или CSV-файл, содержащий тексты.

Если пользователь решил загрузить текст, то происходит отправка POST-запроса по пути <URL сервера веб-приложения>/predictByText. Отправленный текст сначала предобрабатывается с помощью сохраненного препроцессора, а затем классифицируется с помощью загруженной в ОЗУ модели. После этого пользователю отображается результат классификации.

Если пользователь решил загрузить CSV-файл, то происходит отправка POST-запроса по пути <URL сервера веб-приложения>/predictByFile.

К отправленному CSV-файлу предъявляются следующие требования:

- первый столбец имеет название «text» (текст для классификации);
- столбец «text» в каждой клетке содержит текст, который необходимо классифицировать;

- разделителями значений являются «,».

Отправленный CSV-файл сначала предобрабатывается с помощью сохраненного препроцессора, а затем классифицируется с помощью загруженной в ОЗУ модели. После этого пользователю отображаются результаты классификации.

5.5.4 Разработка пользовательского интерфейса

Интерфейс пользователя является одним из главных потребительских свойств программного продукта, поэтому его разработка является важным этапом создания программы.

В рамках проекта требуется разработать интерфейс для прецедентов, описанных в 5.1.

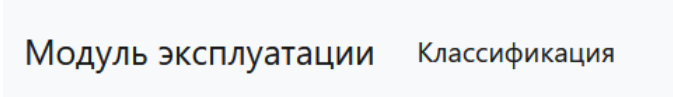
Исходя из указанных требований было решено разработать 3 страницы:

- главная страница;
- страница для отображения результата классификации текста;
- страница для отображения результатов классификации текста из CSV-файла;
- страница, выводящая сообщения о возникающих ошибках.

Для реализации интерфейса используются HTML-страницы и CSS (Cascade Style Sheets)-файлы.

Рассмотрим их разработку более подробно.

Пользователь должен иметь возможность свободно перемещаться между страницами веб-приложения, в связи с чем было принято решение на каждой из них расположить панель навигации, которая содержит ссылку на главную страницу. Ниже на рисунке 12 приведен вид панели навигации.



Модуль эксплуатации Классификация

Рисунок 12 – Панель навигации

Главная страница должна позволять загружать как сам текст, так и CSV-файл, содержащий тексты, а затем выводить результаты классификации. Ниже на рисунке 13 приведен вид главной страницы.

Модуль эксплуатации Классификация

Фрагмент

Классифицировать

Файл

Выбрать... Файл не выбран.

Классифицировать

Рисунок 13 – Главная страница

Страница с результатом классификации текста должна содержать предсказанный класс, точность предсказания и сам текст. Ниже на рисунке 14 приведен вид этой страницы.

Модуль эксплуатации Классификация

Результат

Класс: ДОГ
Точность: 96,27%

Фрагмент

Приём на работу и увольнение работников осуществляются на основании Трудового Кодекса Российской Федерации, Федерального Закона «О высшем и послевузовском профессиональном образовании», «Положения о порядке замещения должностей научно-педагогических работников в высшем учебном заведении», утверждённого приказом Министерства образования и науки РФ №4114 от 26.11.2002 г., и настоящего Коллективного договора.

Рисунок 14 – Страница с результатом классификации по тексту

Страница с результатами классификации текстов из CSV-файла должна содержать таблицу со столбцами «Текст», «Класс», «Точность», в которых располагаются текст, предсказанный класс и точность предсказания. Кроме того,

пользователь должен иметь возможность выгрузить результаты классификации в виде CSV-файла. Ниже на рисунке 15 приведен вид этой страницы.

Модуль эксплуатации Классификация		
Результаты Сохранить		
Фрагмент	Класс	Точность
Безлизовый цифровой голографический микроскоп	ЕИТН	84,90%
Основные характеристики Российской Интернет-аудитории	СОЦН	76,04%
Связь когнитивно-стилевых параметров с мотивационно-личностными и индивидуально-типическими особенностями у мужчин и женщин	СОЦН	80,53%

Рисунок 15 – Страница с результатами классификации по CSV-файлу

Страница, выводящая сообщения о возникающих ошибках должна содержать сообщение о произошедшей ошибке. Ниже на рисунке 16 приведен вид данной страницы.

Модуль эксплуатации Классификация
Ошибка
На вход подан некорректный текст

Рисунок 16 – Страница, выводящая сообщения о возникающих ошибках

6 ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ

6.1 Описание применения

6.1.1 Назначение программы

Разрабатываемое программное средство предназначено для классификации текстовых данных. Оно позволяет обучать модели, используемые для классификации, а также исследовать влияние предобработки текста на точность классификации текстовых данных.

Программное средство состоит из 3 частей – модуля для предобработки текстовых данных, приложения для обучения моделей и веб-приложения для использования обученных моделей.

6.1.2 Условия применения

Так как обучение моделей происходит в Google Colab, то к компьютеру, работающему в этой среде, предъявляются следующие аппаратные требования:

- объем оперативной памяти не менее 4 ГБ;
- центральный процессор с тактовой частотой не менее 2.5 ГГц с поддержкой инструкций SSE3;
- сетевая карта;
- монитор;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру для обучения моделей:

- операционная система Windows 7 или новее (64-бит) или Ubuntu 18.04 или новее (64-бит);
- Google Chrome 80 или новее.

К компьютеру, на котором запущен сервер веб-приложения, предъявляются следующие аппаратные требования:

- объем оперативной памяти не менее 8 ГБ;
- объем жесткого диска не менее 100 ГБ;

- центральный процессор с тактовой частотой не менее 2.5 ГГц с поддержкой инструкций AVX;
- сетевая карта;
- монитор;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру, на котором запущено веб-приложение:

- операционная система Windows 7 или новее (64-бит) или Ubuntu 16.04 или новее (64-бит);
- Python 3.8;

К компьютеру, который использует веб-приложение, предъявляются следующие аппаратные требования:

- центральный процессор с тактовой частотой не менее 2 ГГц;
- сетевая карта;
- монитор;
- встроенная в процессор или дискретная видеокарта;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру, который использует веб-приложение:

- операционная система Windows 7 или новее или Ubuntu 14.04 или новее (64-бит);
- браузер Google Chrome 55.0.2883 или новее или Mozilla Firefox 43 или новее.

На вход приложению для обучения моделей поступает CSV-файл, содержащий столбцы «text» и «class», в которых содержатся тексты и классы, к которым они относятся, соответственно. CSV-файл должен располагаться в

Google Drive по пути «/Colab Notebooks/Text Classification/Datasets/train.csv». Разделителем между значениями должен быть символ «,». На выходе приложения ожидается обученная модель, которая будет сохранена в Google Drive по пути «/Colab Notebooks/Text Classification/Models/trained_model.keras». Так же после обучения модели в Google Colab будет отображена оценка качества обучения модели, включающая в себя:

- изображение графиков значения потерь на обучающей и валидационной выборках;
- изображение графиков точности на обучающей и валидационной выборках;
- изображение графиков чувствительности на обучающей и валидационной выборках;
- изображение графиков специфичности на обучающей и валидационных выборках;
- значение потерь, точность, чувствительность и специфичность на обучающей, валидационной и тестовой выборках.

Входной информацией для веб-приложения является текст или CSV-файл содержащий тексты, выходной информацией – результаты классификации.

6.1.3 Описание задачи

Необходимо позволить пользователю классифицировать текстовые данные, а также исследовать влияние различных методов предобработки текста на точность классификации с помощью нейронных сетей. Для решения данной задачи используются модели, основанные на рекуррентных нейронных сетях.

Чтобы каждый пользователь имел возможность использовать модель для классификации текстовых данных было разработано веб-приложение.

6.1.4 Входные и выходные данные

Входными данными приложения для обучения моделей является файл формата CSV (Comma Separated Values – файл со значениями, разделенными запятыми), соответствующий следующим критериям:

- первый столбец имеет название «text» (текст для классификации);
- столбец «text» в каждой клетке содержит текст, который необходимо классифицировать;
- второй столбец имеет название «class» (класс, к которому относится соответствующий текст);
- столбец «class» в каждой клетке содержит класс, к которому относится соответствующий текст;
- разделителем значений является «,».

CSV-файл содержит все данные, необходимые для обучения модели. Файл должен располагаться в Google Drive по пути «/Colab Notebooks/Text Classification/Datasets/train.csv».

На выходе ожидаются:

- файл обученной модели с названием `trained_model.keras`;
- изображение графиков значения потерь на обучающей и валидационной выборках;
- изображение графиков точности на обучающей и валидационной выборках;
- изображение графиков чувствительности на обучающей и валидационной выборках;
- изображение графиков специфичности на обучающей и валидационных выборках;
- значение потерь, точность, чувствительность и специфичность на обучающей, валидационной и тестовой выборках.

Файл обученной модели сохраняется в Google Drive по пути «/Colab Notebooks/Text Classification/Models/trained_model.keras». В файле содержится обученная модель, использованный препроцессор, который содержит в себе необходимый токенизатор, и, если использовался метод векторизации `word2vec`, матрица весов для слоя `Embedding`.

Входной информацией для веб-приложения является текст или CSV-файл с текстами, выходной информацией – результаты классификации.

6.2 Руководство оператора

6.2.1 Назначение программы

Разработанное программное средство предназначено для классификации текстовых данных и исследования влияния методов предобработки текста на точность классификации. Оно позволяет обучать модели для классификации и использовать их в дальнейшем.

6.2.2 Условия выполнения программы

Программное средство делится на 3 части: модуль предобработки текста, приложение для обучения моделей и веб-приложение, использующее эти модели.

К компьютеру, который использует веб-приложение, предъявляются следующие аппаратные требования:

- центральный процессор с тактовой частотой не менее 2 ГГц;
- сетевая карта;
- монитор;
- встроенная в процессор или дискретная видеокарта;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру, который использует веб-приложение:

- операционная система Windows 7 или новее или Ubuntu 14.04 или новее (64-бит);
- браузер Google Chrome 55.0.2883 или новее или Mozilla Firefox 43 или новее.

К компьютеру, работающему в среде Google Colab для обучения моделей, предъявляются следующие аппаратные требования:

- объем оперативной памяти не менее 4 ГБ;

- центральный процессор с тактовой частотой не менее 2.5 ГГц с поддержкой инструкций SSE3;
- сетевая карта;
- монитор;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

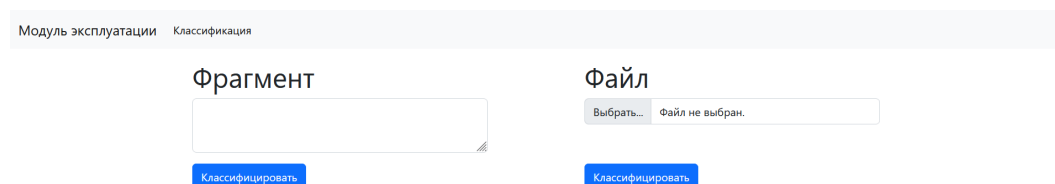
Программные требования к компьютеру для обучения моделей:

- операционная система Windows 7 или новее (64-бит) или Ubuntu 18.04 или новее (64-бит);
- Google Chrome 80 или новее.

6.2.3 Выполнение программы

Веб-приложение

Для доступа к веб-приложению необходимо открыть браузер, соответствующий предъявляемым программным требованиям и ввести в его адресной строке адрес, выдаваемый системным администратором, после чего нажать Enter. Ниже на рисунке 17 приведен результат выполнения данного шага.



The screenshot shows a web application interface with a light gray header containing the text 'Модуль эксплуатации' and 'Классификация'. Below the header, there are two main sections. The left section is titled 'Фрагмент' and contains a large text input field with a small icon in the bottom right corner. Below this field is a blue button labeled 'Классифицировать'. The right section is titled 'Файл' and contains a file selection interface with a 'Выбрать...' button and the text 'Файл не выбран.' Below this is another blue button labeled 'Классифицировать'.

Рисунок 17 – Главная страница

Навигация по приложению осуществляется с помощью навигационной панели, расположенной сверху. На ней находится 1 кнопка – «Классификация», при нажатии на которую открывается главная страница.

На этой странице можно выбрать режим классификации – по тексту или по текстам из CSV-файла.

Для классификации по тексту необходимо заполнить поле под заголовком «Фрагмент» и нажать на кнопку «Классифицировать» (см. рисунок 18).

The screenshot shows a web interface with a header bar containing 'Модуль эксплуатации' and 'Классификация'. Below the header, there are two main sections. The left section is titled 'Фрагмент' and contains a large text input field. A red rectangle highlights this input field, and a red arrow points from the 'Классифицировать' button below it to the input field. The right section is titled 'Файл' and contains a 'Выбрать...' button and a status indicator 'Файл не выбран.' Below the 'Файл' section is another 'Классифицировать' button.

Рисунок 18 – Классификация по тексту

После нажатия на кнопку произойдет переход на страницу с результатом классификации (рисунок 19).

The screenshot shows a web interface with a header bar containing 'Модуль эксплуатации' and 'Классификация'. Below the header, there is a section titled 'Результат'. Under this title, the following information is displayed: 'Класс: ДОГ', 'Точность: 96,27%', and 'Фрагмент'. Below the 'Фрагмент' label, there is a detailed text description: 'Приём на работу и увольнение работников осуществляются на основании Трудового Кодекса Российской Федерации, Федерального Закона «О высшем и послевузовском профессиональном образовании», «Положения о порядке замещения должностей научно-педагогических работников в высшем учебном заведении», утверждённого приказом Министерства образования и науки РФ №4114 от 26.11.2002 г., и настоящего Коллективного договора.'

Рисунок 19 – Результат классификации по тексту

Чтобы вернуться на главную страницу, необходимо нажать на кнопку «Классификация» в навигационной панели.

Для классификации по текстам из CSV-файла необходимо нажать на кнопку «Выбрать...», находящуюся под заголовком «Файл» (рисунок 20).

Рисунок 20 – Открытие диалога для выбора CSV-файла

После этого необходимо выбрать нужный CSV-файл и нажать на кнопку «Открыть».

Название выбранного файла отобразится рядом с кнопкой «Выбрать...». После выбора файла необходимо нажать на кнопку «Классифицировать».

После этого произойдет переход на страницу с результатами классификации. На этой странице отображена таблица с 3 столбцами – «Текст», «Класс» и «Точность», которые соответствуют тексту, предполагаемому классу и точности предположения (рисунок 21).

Фрагмент	Класс	Точность
Безлинзовый цифровой голографический микроскоп	ЕИТН	84,90%
Основные характеристики Российской Интернет-аудитории	СОЦН	76,04%
Связь когнитивно-стилевых параметров с мотивационно-личностными и индивидуально-типическими особенностями у мужчин и женщин	СОЦН	80,53%

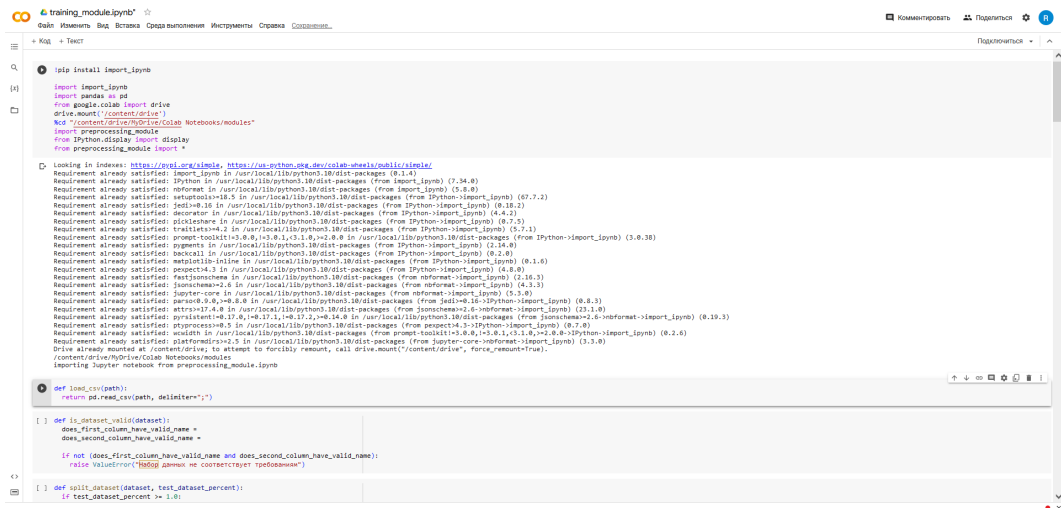
Рисунок 21 – Результат классификации по CSV-файлу

Полученные результаты можно выгрузить в виде CSV-файла, нажав на кнопку «Сохранить».

Приложение для обучения моделей

Файл с приложением для обучения моделей называется `training_module.ipynb`.

Чтобы использовать приложение для обучения моделей необходимо открыть файл `training_module.ipynb` в Google Colab (рисунок 22). Он должен быть расположен по пути `«/Colab Notebooks/Text Classification/Modules/training_module.ipynb»`.



```
import_ipynb
import pandas as pd
from google.colab import drive
drive.mount('/content/drive')
!pip install --target=/content/drive/Colab Notebooks/modules/ preprocessing_module
from IPython.display import display
from preprocessing_module import *
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: import_ipynb in /usr/local/lib/python3.10/dist-packages (0.1.4)
Requirement already satisfied: IPython in /usr/local/lib/python3.10/dist-packages (from import_ipynb) (7.34.0)
Requirement already satisfied: notebook in /usr/local/lib/python3.10/dist-packages (from import_ipynb) (5.8.0)
Requirement already satisfied: setuptools>=18.5 in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (67.7.2)
Requirement already satisfied: jupyterlab in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (0.18.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (0.7.5)
Requirement already satisfied: traitlets>=4.2 in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (5.7.1)
Requirement already satisfied: backcall in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (0.2.0)
Requirement already satisfied: pygments in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (2.14.0)
Requirement already satisfied: pexpect in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (4.8.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (0.1.6)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.10/dist-packages (from IPython->import_ipynb) (1.5.1)
Requirement already satisfied: FastJSONSchema in /usr/local/lib/python3.10/dist-packages (from nbformat->import_ipynb) (2.16.3)
Requirement already satisfied: jsonschema>=2.6 in /usr/local/lib/python3.10/dist-packages (from nbformat->import_ipynb) (4.17.3)
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.10/dist-packages (from nbformat->import_ipynb) (5.3.0)
Requirement already satisfied: parso>=0.8.0<0.9 in /usr/local/lib/python3.10/dist-packages (from jedi>=0.18.0->IPython->import_ipynb) (0.8.3)
Requirement already satisfied: attrs>=17.4.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->import_ipynb) (22.1.0)
Requirement already satisfied: pyrsistent>=0.17.0, <0.17.1, >=0.17.2, <0.18.0 in /usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6->nbformat->import_ipynb) (0.19.3)
Requirement already satisfied: pyprocess in /usr/local/lib/python3.10/dist-packages (from perspective>4.3->IPython->import_ipynb) (0.7.8)
Requirement already satisfied: distutils in /usr/local/lib/python3.10/dist-packages (from prompt-toolkit>=3.0.0, <3.0.1, >3.0.1, <3.0.2, >3.0.0->IPython->import_ipynb) (0.2.0)
Requirement already satisfied: platformdirs>=2.3 in /usr/local/lib/python3.10/dist-packages (from jupyter-core->nbformat->import_ipynb) (3.3.0)
Drive already mounted at /content/drive, to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
/content/drive/MyDrive/Colab Notebooks/modules
Importing Jupyter notebook from preprocessing_module.ipynb
```

```
def load_csv(path):
    return pd.read_csv(path, delimiter=',')
```

```
[ ] def is_dataset_valid(dataset):
    does_first_column_have_valid_name =
    does_second_column_have_valid_name =
    if not (does_first_column_have_valid_name and does_second_column_have_valid_name):
        raise ValueError("Набор данных не соответствует требованиям")

[ ] def split_dataset(dataset, test_dataset_percent):
    if test_dataset_percent >= 1.0:
```

Рисунок 1 – Приложение для обучения моделей

Для обучения модели необходимо расположить набор данных в виде CSV-файла по пути `«/Colab Notebooks/Text Classification/Datasets/train.csv»`.

К CSV-файлу предъявляются следующие требования:

- первый столбец имеет название «text» (текст для классификации);
- столбец «text» в каждой клетке содержит текст, который необходимо классифицировать;
- второй столбец имеет название «class» (класс, к которому относится соответствующий текст);
- столбец «class» в каждой клетке содержит класс, к которому относится соответствующий текст;
- разделителями значений являются «,».

Задать параметры для препроцессора можно в специальной ячейке под названием «Настройка препроцессора» (рисунок 23)

▼ Настройка препроцессора

```
[ ] def get_preprocessor():  
    # Создадим препроцессор  
    preprocessor = Preprocessor()  
  
    preprocessor.add_preprocessing(SequenceVectorizationMechanism(SequenceVectorizationMode.WORD2VEC_CBOU, {"num_words": 3000, "max_sequence_length": 500, "window_size": 3, "vector_size": 100}))
```

Рисунок 23 – Задание параметров для препроцессора

Для добавления определенной предобработки необходимо добавить следующую строку (см. таблицу 12 для нахождения нужных значений подстановок):

```
preprocessor.add_preprocessing(<название механизма>(<название перечисления с режимами работы механизма>.<значение перечисления с режимами работы механизма>, <дополнительные параметры>))
```

Таблица 12 – Параметры для добавления метода предобработки

Название механизма	Название перечисления с режимами работы механизма	Значение перечисления с режимами работы механизма	Метод предобработки
SimpleDatasetPreprocessing Mechanism	SimpleDatasetPreprocessing	DELETE_DUPLICATES	Удаление дубликатов
SimpleTextPreprocessing Mechanism	SimpleTextPreprocessing	DELETE_PUNCTUATION	Удаление символов пунктуации
		DELETE_NUMBERS	Удаление чисел
		DELETE_EXTRA_WHITES PACES	Удаление лишних пробельных символов
		TO_LOWER_CASE	Приведение слов к нижнему буквенному регистру
		DELETE_STOPWORDS	Удаление стоп-слов
WordNormalization Mechanism	WordNormalizationMode	STEMMING	Стемминг
		LEMMATIZATION	Лемматизация
SequenceVectorizationMechanism	SequenceVectorizationMode	BAG_OF_WORDS	Мешок слов
		TF_IDF	TF-IDF
		WORD2VEC_CBOW	Word2Vec CBOW
		WORD2VEC_SKIP_GRAM	Word2Vec Skip Gram
DatasetBalancing Mechanism	DatasetBalancingMode	DOWNSAMPLING	Апсемплинг
		UPSAMPLING	Даунсемплинг
		AVERAGING	Приведение количества примеров в каждом классе к среднему значению

Для некоторых из методов предобработок можно задать дополнительные параметры (подстановка <дополнительные параметры>). Эти параметры описаны в таблице 13. Если метод предобработки не имеет дополнительных

параметров, то на месте подстановки <дополнительные параметры> необходимо указать «{ }». Пример значения для подстановки <дополнительные параметры> - «{"num_words": 1000, "max_sequence_length": 1000, "window_size": 3, "vector_size": 100}».

Таблица 23 – Дополнительные параметры при добавлении метода предобработки

Метод предобработки	Дополнительный параметр	Описание дополнительного параметра	Значение по умолчанию	Пример значения дополнительного параметра
Удаление стоп-слов	language	Название языка для которого будет загружен список стоп-слов.	russian	russian
				english
Стемминг	language	Название языка для которого будет проводиться стемминг	russian	russian
				english
Лемматизация	language	Название языка для которого будет проводиться лемматизация	russian	russian
				english
Мешок слов	num_words	Размер словаря	1000	1000
				2000
TF-IDF	num_words	Размер словаря	1000	1000
				2000
Word2Vec CBOW / Skip Gram	num_words	Размер словаря	1000	1000
				2000
	max_sequence_length	Длина векторизованного текста	1000	1000
				500
	windows_size	Размер окна	3	3
				5
	vector_size	Размер эмбединга	100	100
				200

После этого для запуска обучения модели необходимо нажать на кнопку запуска соответствующей ячейки (рисунок 24).

Обучение модели

```

model = get_model(preprocessor, sequences, labels)
epochs = 100
batch_size = 32
train_model(dataset, preprocessor, model, epochs, batch_size)

```

Рисунок 24 – Ячейка обучения модели

После окончания обучения будет выведена оценка качества обученной модели, а снизу появится зеленая галочка (рисунок ...).

рисунок

В результате обучения по пути «/Colab Notebooks/Text Classification/Models/trained_model.keras» будет сохранена обученная модель, которую в дальнейшем можно использовать в веб-приложении (развертыванием модели занимается системный администратор).

6.2.4 Сообщения оператору

Веб-приложение

В ходе работы с веб-приложением оператору могут быть выданы различные сообщения.

При попытке классифицировать пустую строку будет выдано сообщение «На вход подан некорректный текст». При его возникновении необходимо нажать на кнопку «Классификация» в навигационной панели и ввести корректный текст.

При попытке загрузить CSV-файл, не удовлетворяющий вышеперечисленным требованиям будет выдано сообщение «Загруженный CSV-файл не соответствует предъявленным требованиям». При его возникновении необходимо нажать на кнопку «Классификация» в навигационной панели и загрузить CSV-файл, удовлетворяющий вышеперечисленным требованиям.

Приложение для обучения моделей

Во время работы с приложением для обучения моделей могут быть выведены разные сообщения.

При успешной загрузке набора данных выводится сообщение «Общее количество примеров: <количество примеров в наборе данных>», а также отображается таблица с количеством примеров в каждом классе.

Если в ходе загрузки набора данных произошла ошибка, то выводится сообщение «Не удалось загрузить набор данных». При его возникновении необходимо проверить подаваемый на вход CSV-файл.

Если загружаемый CSV-файл не прошел проверку на соответствие требованиям, то выводится сообщение «CSV-файл не соответствует заявленным требованиям». При его возникновении необходимо проверить подаваемый на вход CSV-файл.

Если во время обучения произошла ошибка, то выводится сообщение об ошибке «Во время обучения произошла ошибка». При его возникновении необходимо перезапустить приложение. В случае повторного появления – связаться с системным администратором.

Если во время сохранения модели произошла ошибка, то выводится сообщение «Не удалось сохранить модель». При его возникновении необходимо связаться с системным администратором.

При успешном завершении приложения выводится сообщение «Обучение модели было успешно завершено».

6.3 Руководство программиста

6.3.1 Назначение и условия применения программы

Разработанное программное средство предназначено для классификации текстовых данных и исследования влияния методов предобработки текста на точность классификации. Оно позволяет обучать модели для классификации и использовать их в дальнейшем.

Программное средство делится на 3 части: модуль предобработки текста, приложение для обучения моделей и веб-приложение, использующее эти модели.

К компьютеру, который использует веб-приложение, предъявляются следующие аппаратные требования:

- центральный процессор с тактовой частотой не менее 2 ГГц;
- сетевая карта;
- монитор;
- встроенная в процессор или дискретная видеокарта;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру, который использует веб-приложение:

- операционная система Windows 7 или новее или Ubuntu 14.04 или новее (64-бит);
- браузер Google Chrome 55.0.2883 или новее или Mozilla Firefox 43 или новее.

К компьютеру, работающему в среде Google Colab для обучения моделей, предъявляются следующие аппаратные требования:

- объем оперативной памяти не менее 4 ГБ;
- центральный процессор с тактовой частотой не менее 2.5 ГГц с поддержкой инструкций SSE3;
- сетевая карта;
- монитор;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру для обучения моделей:

- операционная система Windows 7 или новее (64-бит) или Ubuntu 18.04 или новее (64-бит);
- Google Chrome 80 или новее.

6.3.2 Обращения к программе

Обращение к приложению для обучения моделей происходит в Google Colab через запуск соответствующих ячеек кода.

Обращение к веб-приложению происходит при помощи HTTP-запросов, а именно GET- и POST-запросов (таблица 14).

Таблица 14 – Возможные HTTP-запросы к веб-приложению

Метод запроса	URL	Параметры запроса	Описание
GET	<URL сервера веб-приложения>	Отсутствуют	Возвращает HTML-страницу с формами загрузки текста для классификации
POST	<URL сервера веб-приложения>/predictByText	text – загружаемый текст	Производит загрузку текста text, его классификацию и возвращает HTML-страницу с результатом классификации
POST	<URL сервера веб-приложения>/predictByFile	file – загружаемый CSV-файл с текстами	Производит загрузку CSV-файла с текстами, классифицирует тексты и возвращает HTML-страницу с результатами классификации

6.3.3 Входные и выходные данные

Входные и выходные данные приложения для обучения моделей и веб-приложения были описаны в 6.1.4.

Выходными данными веб-приложения с точки зрения программиста являются HTML-страницы с соответствующим содержимым.

6.3.4 Сообщения

Сообщения были описаны в 6.2.4.

6.4 Руководство системного программиста

6.4.1 Общие сведения о программе

Разработанное программное средство предназначено для классификации текстовых данных и исследования влияния методов предобработки текста на точность классификации. Оно позволяет обучать модели для классификации и использовать их в дальнейшем.

Программное средство делится на **2 части**: приложение для обучения моделей и веб-приложение, использующее эти модели.

К компьютеру, который использует веб-приложение, предъявляются следующие аппаратные требования:

- центральный процессор с тактовой частотой не менее 2 ГГц;
- сетевая карта;
- монитор;
- встроенная в процессор или дискретная видеокарта;

- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру, который использует веб-приложение:

- операционная система Windows 7 или новее или Ubuntu 14.04 или новее (64-бит);
- браузер Google Chrome 55.0.2883 или новее или Mozilla Firefox 43 или новее.

К компьютеру, работающему в среде Google Colab для обучения моделей, предъявляются следующие аппаратные требования:

- объем оперативной памяти не менее 4 ГБ;
- центральный процессор с тактовой частотой не менее 2.5 ГГц с поддержкой инструкций SSE3;
- сетевая карта;
- монитор;
- манипулятор типа мышь, клавиатура или подобное указывающее устройство.

Программные требования к компьютеру для обучения моделей:

- операционная система Windows 7 или новее (64-бит) или Ubuntu 18.04 или новее (64-бит);
- Google Chrome 80 или новее.

6.4.2 Структура программы

Приложение для обучения моделей состоит из файла `training_module.ipynb`. Само приложение представляет из себя ноутбук Jupyter Notebook, запускаемый в среде Google Colab.

Приложение для обучения моделей и веб-приложение используют модуль предобработки текста, представленный в виде файла `preprocessing_module.py`.

Приложение для обучения моделей (`training_module.ipynb`) и модуль предобработки текста должны находиться по следующему пути в Google Drive - «/Colab Notebooks/Text Classification/Modules».

Сервер веб-приложения запускается при помощи интерпретатора языка Python.

6.4.3 Настройка программы

Запуск веб-приложения на сервере производится следующим образом.

1. В директории с веб-приложением необходимо разместить обученную модель `trained_model.keras`.
2. В директории с веб-приложением выполнить команду «`python3 manage.py runserver`» для запуска веб-приложения.

6.4.4 Проверка программы

Чтобы проверить работоспособность приложения для обучения моделей необходимо произвести обучение хотя бы одной модели. Если приложение при завершении своей работы выдает сообщение «Обучение модели было успешно завершено», значит оно работоспособно.

Чтобы проверить работоспособность веб-приложения необходимо в браузере перейти по адресу сервера веб-приложений. При этом должна открыться главная страница, на которой будет предложено загрузить текст для классификации. Попробуйте загрузить текст. В случае работоспособности будет выведен результат классификации.

6.4.5 Сообщения системному программисту

Сообщения были описаны в 6.2.4.

Если во время запуска веб-приложения произошла ошибка загрузки обученной модели, то будет выведено сообщение «Не удалось загрузить модель». При его возникновении необходимо проверить наличие модели в папке с приложением и ее целостность.

7 ТЕСТИРОВАНИЕ

7.1 Исследование влияния методов предобработки текста на точность классификации текстовых данных

Для исследования влияния методов предобработки текста на точность классификации текстовых данных был выбран набор данных, состоящий из фрагментов коллективных договоров. Всего в наборе данных 94751 примеров, 8 классов.

В качестве модели использовалась простейшая LSTM-сеть:

```
def get_model(preprocessor, tensors, labels):
    model=Sequential()
    model.add(Bidirectional(LSTM(32, return_sequences=False), input_shape=tensors.shape))
    model.add(Dropout(0.3))
    model.add(Dense(labels.shape[1], activation='softmax'))
    model.compile(optimizer='adam',
                  loss='categorical_crossentropy',
                  metrics=['accuracy',tf.keras.metrics.Recall(name="sensitivity"),specificity])

    return model
```

Сначала была произведена оценка качества модели при использовании непредобработанного набора данных. Поскольку для подачи на вход нейронной сети текст должен быть векторизован, а набор данных - сбалансирован, то в качестве метода векторизации был выбран метод "Мешок слов", а в качестве метода балансирования набора данных - "Приведение количества примеров в каждом классе к среднему значению".

Размер батча равен 128. Максимальное количество эпох - 100. Обучение заканчивается как только значение потерь на валидационном наборе данных перестает уменьшаться.

Точность составила 61,1%, чувствительность – 59,9%, специфичность – 94,8%.

Рассмотрим влияние каждой группы методов предобработки текстовых данных подробнее.

Простые методы предобработки текста

Метод векторизации и метод балансирования набора данных соответствуют тем, которые применялись для непредобработанного набора

данных. Обучение производилось с тем же размер батча и максимальным количеством эпох.

Ниже в таблице 15 приведены результаты исследования влияния простых методов предобработки текста на точность классификации текстовых данных.

Таблица 15 – Влияние простых методов предобработки текста

Метод предобработки	Точность	Чувствительность	Специфичность
Удаление чисел	59,7%	58,0%	94,3%
Приведение слов к нижнему буквенному регистру	62,3%	60,7%	95,0%
Удаление излишних пробельных символов	61,7%	60,6%	94,8%
Удаление стоп-слов	62,4%	62,5%	94,9%
Удаление символов пунктуации	62,2%	61,8%	94,9%
Все простые методы предобработки текста, кроме удаления чисел	64,3%	62,8%	95,1%
Все простые методы предобработки текста	62,3%	60,9%	94,8%

Использование простых методы предобработки позволило повысить точность модели на ~3% по сравнению с использованием непредобработанного набора данных. Стоит отметить, что удаление чисел понизило точность модели, из чего можно сделать предположение, что используемые во фрагментах коллективного договора числа играют роль в определении раздела.

Простые методы предобработки набора данных

Метод векторизации и метод балансирования набора данных соответствуют тем, которые применялись для непредобработанного набора данных. Обучение производилось с тем же размер батча и максимальным количеством эпох.

Ниже в таблице 16 приведены результаты исследования влияния простых методов предобработки набора данных на точность классификации текстовых данных.

Таблица 16 – Влияние простых методов предобработки набора данных

Метод предобработки	Точность	Чувствительность	Специфичность
Удаление дубликатов	60,8%	59,6%	94,7%

Удаление дубликатов в данном случае не оказало значимого влияния на точность. Из этого можно сделать предположение, что набор данных не обладает большим количеством дублированных примеров.

Методы нормализации слов

Метод векторизации и метод балансирования набора данных соответствуют тем, которые применялись для непредобработанного набора данных. Обучение производилось с тем же размер батча и максимальным количеством эпох.

Ниже в таблице 17 приведены результаты исследования влияния методов нормализации слов на точность классификации текстовых данных.

Таблица 17 – Влияние простых методов предобработки набора данных

Метод предобработки	Точность	Чувствительность	Специфичность
Стемминг	65,6%	65,2%	95,5%
Лемматизация	66,5%	66,4%	95,6%

Применение методов нормализации слов позволило увеличить точность модели на ~5% по сравнению с использованием непредобработанного набора данных. Лемматизация оказала большее влияние на точность, чем стемминг.

Методы векторизации текста

Метод балансирования набора данных соответствует тому, который применялся для непредобработанного набора данных. Обучение производилось с тем же размер батча и максимальным количеством эпох.

Для методов векторизации word2vec модель была модифицирована для поддержки слоя Embedding:

```
def get_model(preprocessor, tensors, labels):
    embedding_matrix = preprocessor.output_params["embedding_matrix"]
    model = Sequential()
    model.add(Embedding(embedding_matrix.shape[0], embedding_matrix.shape[1], mask_zero=False,
weights=[embedding_matrix], input_length=tensors.shape[1], trainable=False))
    model.add(LSTM(32))
```

```

model.add(Dropout(0.3))
model.add(Dense(labels.shape[1], activation='softmax'))
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy', tf.keras.metrics.Recall(name="sensitivity"),
specificity])

return model

```

Ниже в таблице 18 приведены результаты исследования влияния методов векторизации текста на точность классификации текстовых данных.

Таблица 18 – Влияние простых методов векторизации текста

Метод предобработки	Точность	Чувствительность	Специфичность
Мешок слов	61,1%	59,9%	94,8%
TF-IDF	63,3%	63,0%	95,1%
Word2Vec CBOW	63,7%	63,3%	95,3%
Word2Vec Skip Gram	63,8%	63,3%	95,4%

Применение методов нормализации слов позволило увеличить точность модели на ~3% по сравнению с использованием непредобработанного набора данных. Из всех методов векторизации текста Word2Vec Skip Gram оказал наибольшее влияние на точность.

Методы балансирования текстового набора данных

Метод векторизации соответствует тому, который применялся для непредобработанного набора данных. Обучение производилось с тем же размер батча и максимальным количеством эпох.

Ниже в таблице 19 приведены результаты исследования влияния методов балансирования текстового набора данных на точность классификации текстовых данных.

Таблица 19 – Влияние методов балансирования текстового набора данных

Метод предобработки	Точность	Чувствительность	Специфичность
Даунсемплинг	60,7%	56,6%	93,8%
Апсемплинг	61,8%	60,3%	94,9%
Приведение количества примеров в каждом классе к среднему значению	61,1%	59,9%	94,8%

Апсемплинг оказал наибольшее влияние на точность.

Рассмотрев каждую из групп методов предобработки текстовых данных, было решено применить те, которые оказали наибольшее влияние на точность, вместе.

Были использованы следующие методы предобработки:

- удаление символов пунктуации;
- удаление излишних пробельных символов;
- приведение слов к нижнему буквенному регистру;
- удаление стоп-слов;
- лемматизация;
- word2vec Skip Gram;
- апсемплинг.

В результате точность составила 69,7%, чувствительность – 68,3%, специфичность – 95,9%.

Таким образом, применение различных методов предобработки текста позволило повысить точность модели на ~9% по сравнению с использованием непредобработанного текстового набора данных.

Стоит отметить, что при использовании более продвинутых моделей влияние может оказаться иным. Более того, как было сказано в разделе 4.3.3, применение большего набора данных потенциально может улучшить качество эмбедингов, что в свою очередь позволит повысить точность.

7.2 Приложение для обучения моделей

7.2.1 План тестирования

В ходе тестирования приложения для обучения моделей необходимо проверить следующие ситуации:

- загрузка набора данных при наличии по указанному пути нужных данных;
- загрузка набора данных при отсутствии по указанному пути нужных данных;

- обучение модели с препроцессором без добавления механизмов предобработки;
- обучение модели с препроцессором с добавлением механизмов предобработки;
- запуск исследования влияния предобработки текста на точность классификации текстовых данных.

7.2.2 Результаты тестирования

Загрузка набора данных при наличии по указанному пути нужных данных

Приложение запущено, производится попытка загрузить данные по указанному пути.

Результат: набор данных успешно загружен.

Загрузка набора данных при отсутствии по указанному пути нужных данных

Приложение запущено, производится попытка загрузить данные по указанному пути.

Результат: выводится сообщение об ошибке «Не удалось найти набор данных».

Обучение модели с препроцессором без добавления механизмов предобработки

Приложение запущено, набор данных загружен, в препроцессор не добавлены механизмы предобработки, производится обучение модели.

Результат: модель успешно обучена.

Обучение модели с препроцессором с добавлением механизмов предобработки

Приложение запущено, набор данных загружен, в препроцессор добавлены механизмы предобработки, модель обучена, производится обучение модели.

Результат: модель успешно обучена.

Запуск исследования влияния предобработки текста на точность классификации текстовых данных

Приложение запущено, набор данных загружен, запускается метод исследования влияния предобработки текста на точность классификации текстовых данных.

Результат: выводится таблица с результатами тестирования моделей, обученных на наборах данных, предобработанных различными методами.

7.2.3 Анализ результатов тестирования

В ходе тестирования приложения для обучения моделей оно корректно выполнило все запланированные сценарии и в случае исключительных ситуаций вывела соответствующие сообщения об ошибках.

7.3 Веб-приложение

7.3.1 План тестирования

В ходе тестирования веб-приложения необходимо проверить следующие ситуации:

- пользователь загружает корректный текст для классификации;
- пользователь загружает некорректный текст для классификации;
- пользователь загружает CSV-файл, соответствующий вышеизложенным требованиям для классификации;
- пользователь загружает CSV-файл, не соответствующий вышеизложенным требованиям для классификации;
- приложение запускается, обученная модель найдена;
- приложение запускается, обученная модель не найдена.

7.3.2 Результаты тестирования

Пользователь загружает корректный текст для классификации

Приложение запущено, пользователь открывает главную страницу, вводит корректный текст для классификации и нажимает кнопку «Классифицировать».

Результат: приложение выдает результат классификации текста.

Пользователь загружает некорректный текст для классификации

Приложение запущено, пользователь открывает главную страницу, ничего не вводит и нажимает кнопку «Классифицировать» под заголовком «Фрагмент».

Результат: приложение выдает сообщение об ошибке «На вход подан некорректный текст».

Пользователь загружает CSV-файл, соответствующий вышеизложенным требованиям для классификации

Приложение запущено, пользователь открывает главную страницу, выбирает корректный CSV-файл для классификации и нажимает кнопку «Классифицировать».

Результат: приложение выдает результаты классификации текстов из CSV-файла.

Пользователь загружает CSV-файл, не соответствующий вышеизложенным требованиям для классификации

Приложение запущено, пользователь открывает главную страницу, выбирает некорректный CSV-файл для классификации и нажимает кнопку «Классифицировать».

Результат: приложение выдает сообщение об ошибке «Загруженный CSV-файл не соответствует предъявленным требованиям».

Приложение запускается, обученная модель найдена

Системный администратор размещает обученную модель в папке с приложением, запускает приложение, открывает главную страницу, вводит корректный текст для классификации и нажимает кнопку «Классифицировать».

Результат: приложение выдает результат классификации текста.

Приложение запускается, обученная модель не найдена

Системный администратор убеждается в отсутствии обученной модели в папке с приложением, запускает приложение, открывает главную страницу.

Результат: приложение выводит страницу с сообщением об ошибке.

7.3.3 Анализ результатов тестирования

В ходе тестирования веб-приложения оно корректно выполнило все запланированные сценарии и в случае исключительных ситуаций вывела соответствующие сообщения об ошибках.

ЗАКЛЮЧЕНИЕ

В настоящее время в мире наблюдается стремительный рост объема данных. Не исключением являются и текстовые данные. Однако, чтобы извлечь из них пользу их необходимо проанализировать. Одним из инструментов анализа являются нейронные сети, на точность которых можно повлиять, используя различные методы предобработки текстовых данных. Созданное программное обеспечение позволит производить обработку текстовых данных различными методами и повысить точность классификации текстовых данных.

В выпускной квалификационной работе было разработано программное средство, позволяющее производить предобработку текстовых данных и классифицировать их, состоящее из 3 частей: модуль предобработки текстовых данных, приложение для обучения моделей и веб-приложение для классификации текстовых данных. Был обоснован выбор средств разработки. Разработана архитектура программного обеспечения и алгоритмы функционирования основных частей программы. Сформулированы требования к программному и аппаратному обеспечению. Приведена программная документация, включающая руководство оператора, руководство программиста и системного программиста, необходимые для использования и сопровождения программного продукта.

Можно заключить, что работа по созданию средства для предобработки текстовых данных имеет перспективы и может быть продолжена. Основным ограничивающим фактором является аппаратное обеспечение.

В качестве направлений развития можно отметить использование большего количества данных для синтезирования модели и применение более мощного аппаратного обеспечения, которое бы позволило более полно производить оценку влияния различных методов предобработки текстовых данных на точность их классификации при помощи различных типов нейронных сетей.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Фаулер М. Шаблоны корпоративных приложений. / пер. с. англ. Т.П. Кайгородовой. – М.: Вильямс, 2016. – 544 с.
2. Chollet F. Deep Learning with Python. – Shelter Island: Manning Publications Co., 2018. – 384 p.
3. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. – Sebastopol: O'Reilly Media, 2019. – 856 p.
4. Goodfellow I. et al. Deep Learning. – Cambridge: MIT Press, 2016. – 800 p.
5. Huang J. Accelerating AI with GPUs: A New Computing Model // NVIDIA Blog [сайт]. – URL: <https://blogs.nvidia.com/blog/2016/01/12/accelerating-ai-artificial-intelligence-gpus/> (дата обращения 18.05.2021).
6. Johnson J., Khoshgoftaar T. Survey on deep learning with class imbalance // Journal of Big Data. – 2019. – № 6. – pp. 125-142.
7. Krawczyk B. Learning from imbalanced data: open challenges and future directions // Progress in Artificial Intelligence. – 2016. – № 5, pp. 221–232.
8. The benefits of using web-based applications // Geeks [сайт]. – URL: <https://www.geeks.ltd.uk/about-us/blog/details/eQU5Ip/the-benefits-of-using-web-based-applications> (дата обращения 03.06.2021).
9. VRAM Reserved by Windows 10 // NVIDIA GeForce Forums [сайт]. – URL: <https://www.nvidia.com/en-us/geforce/forums/off-topic/25/260612/vram-reserved-by-windows-10/> (дата обращения: 16.05.2021).

ПРИЛОЖЕНИЕ А. ЛИСТИНГ НАИБОЛЕЕ ЗНАЧИМЫХ ЧАСТЕЙ ПРОГРАММЫ

Модуль предобработки текста (preprocessing_module.py)

```
# Тема ВКР: Разработка программного обеспечения для исследования влияния предобработки текста на
# точность решения задач классификации текстовой информации
# Автор: Головкин Н.В., 09.04.04, 143М
# Научный руководитель: Цуканова Н.И., доцент
# Средства разработки: Python 3.8, Keras 2.12.0
# Назначение данной части ПО: Модуль предобработки текста
# Дата создания: 07.05.2023
import os
# Установим библиотеку для лемматизации
os.popen('pip install pymorphy2').read()
import tensorflow as tf

from enum import Enum

# Keras
from keras.preprocessing.text import Tokenizer
from keras.models import Sequential
from keras.layers import Dense, Flatten, Embedding, LSTM, SpatialDropout1D
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
from keras.callbacks import EarlyStopping
from keras.layers import Dropout
from keras.utils.data_utils import pad_sequences

# Работа с Google Drive
from google.colab import drive
from google.colab import output
from os import listdir

# Работа с естественным языком
import nltk
import pymorphy2
import string
from nltk.stem import SnowballStemmer
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords as sw
from nltk import FreqDist
from gensim.test.utils import common_texts
from gensim.models import Word2Vec

# Прочее
import pickle
import numpy as np;
import pandas as pd;
import matplotlib as mpl
import matplotlib.pyplot as plt
import re
from IPython.display import display, HTML

# Простые методы предобработки текста
class SimpleTextPreprocessing(Enum):
    # Удаление символов пунктуации
    DELETE_PUNCTUATION = 1
    # Удаление чисел
    DELETE_NUMBERS = 2
    # Удаление излишних пробельных символов
    DELETE_EXTRA_WHITESPACES = 3
    # Приведение слов к нижнему буквенному регистру
    TO_LOWER_CASE = 4
    # Удаление стоп-слов
    DELETE_STOPWORDS = 5

# Механизм простой предобработки текста
class SimpleTextPreprocessingMechanism:
    # Инициализация объекта класса SimpleTextPreprocessingMechanism
    # Входные параметры:
    #     simple_text_processing (обязательный): простой метод предобработки текста - одно из значений SimpleTextPreprocessing
    #     params (необязательный): словарь с параметрами для простого метода предобработки текста
    def __init__(self, simple_text_processing, params={"language": "russian"}):
        self.simple_text_processing = simple_text_processing
        self.params = params
```

```

        if not isinstance(self.simple_text_processing, SimpleTextPreprocessing):
            raise ValueError("Укажите в параметре simple_text_processing одно из значений перечисления SimpleTextPreprocessing")
        nltk.download('stopwords')
        self.stopwords = sw.words("russian")
        nltk.download("punkt")
        self.punctuations = list(string.punctuation)
        self.punctuations.append('-')
        self.punctuations.append('«')
        self.punctuations.append('»')
        self.punctuations.append('•')
        self.punctuations.append('\n')
        self.punctuations.append('\r')

# Обработка текста
def process(self, text):
    if self.simple_text_processing == SimpleTextPreprocessing.DELETE_PUNCTUATION:
        return self.delete_punctuation(text, self.params)
    elif self.simple_text_processing == SimpleTextPreprocessing.DELETE_NUMBERS:
        return self.delete_numbers(text, self.params)
    elif self.simple_text_processing == SimpleTextPreprocessing.DELETE_EXTRA_WHITESPACES:
        return self.delete_extra_whitespace(text, self.params)
    elif self.simple_text_processing == SimpleTextPreprocessing.TO_LOWER_CASE:
        return self.to_lower_case(text, self.params)
    elif self.simple_text_processing == SimpleTextPreprocessing.DELETE_STOPWORDS:
        return self.delete_stopwords(text, self.params)
    else:
        return text

# Удаление пунктуации
def delete_punctuation(self, text, params):
    preprocessed_text = text
    for punctuation in self.punctuations:
        preprocessed_text = preprocessed_text.replace(punctuation, '')

    return preprocessed_text

# Удаление чисел
def delete_numbers(self, text, params):
    return re.sub(r"([0-9]+\.\d*)+", "", text)

# Удаление излишних пробелов
def delete_extra_whitespace(self, text, params):
    preprocessed_text = text.strip()
    return re.sub(r"\s{2,}", " ", text)

# Приведение строки к нижнему регистру
def to_lower_case(self, text, params):
    return text.lower()

# Удаление стоп-слов
def delete_stopwords(self, text, params):
    word_tokens = [word_token for word_token in word_tokenize(text, language=params["language"]) if
not word_token in self.stopwords]
    preprocessed_text = ' '.join(word_tokens)

    return preprocessed_text

# Простые методы предобработки набора данных
class SimpleDatasetPreprocessing(Enum):
    # Удаление дубликатов
    DELETE_DUPLICATES = 1

# Механизм простой предобработки набора данных
class SimpleDatasetPreprocessingMechanism:
    # Инициализация объекта класса SimpleDatasetPreprocessingMechanism
    # Входные параметры:
    #     simple_dataset_preprocessing (обязательный): простой метод предобработки набора данных - одно
из значений SimpleDatasetPreprocessing
    #     params (необязательный): словарь с параметрами для простого метода предобработки набора данных
    def __init__(self, simple_dataset_preprocessing, params={}):
        self.simple_dataset_preprocessing = simple_dataset_preprocessing
        self.params = params
        if not isinstance(self.simple_dataset_preprocessing, SimpleDatasetPreprocessing):
            raise ValueError("Укажите в параметре simple_dataset_preprocessing одно из значений
перечисления SimpleDatasetPreprocessing")

# Обработка набора данных
def process(self, dataset):
    if self.simple_dataset_preprocessing == SimpleDatasetPreprocessing.DELETE_DUPLICATES:

```



```

        return self.delete_duplicates(dataset, self.params)
    else:
        return dataset

# Удаление дубликатов
def delete_duplicates(self, dataset, params):
    duplicate_column = params["duplicate_column"]

    return dataset.drop_duplicates(subset=duplicate_column, keep="first")

# Методы нормализации
class WordNormalizationMode(Enum):
    # Стемминг
    STEMMING = 1
    # Лемматизация
    LEMMATIZATION = 2

# Механизм нормализации
class WordNormalizationMechanism:
    # Инициализация объекта класса WordNormalizationMechanism
    # Входные параметры:
    # word_normalization_mode (обязательный): метод нормализации - одно из значений
    WordNormalizationMode
    # params (необязательный): словарь с параметрами для метода нормализации
    def __init__(self, word_normalization_mode, params={"language": "russian"}):
        self.word_normalization_mode = word_normalization_mode
        self.params = params
        if not isinstance(self.word_normalization_mode, WordNormalizationMode):
            raise ValueError("Укажите в параметре word_normalization_mode одно из значений перечисления
WordNormalizationMode")
        if self.word_normalization_mode == WordNormalizationMode.STEMMING:
            self.stemmer = SnowballStemmer(language="russian")
        elif self.word_normalization_mode == WordNormalizationMode.LEMMATIZATION:
            self.lemmatizer = pymorphy2.MorphAnalyzer()

# Обработка текста
def process(self, text):
    if self.word_normalization_mode == WordNormalizationMode.STEMMING:
        return self.stemming(text, self.params)
    elif self.word_normalization_mode == WordNormalizationMode.LEMMATIZATION:
        return self.lemmatization(text, self.params)
    else:
        return text

# Стемминг
def stemming(self, text, params):
    return ' '.join([self.stemmer.stem(word_token) for word_token in word_tokenize(text,
language="russian")])

# Лемматизация
def lemmatization(self, text, params):
    return ' '.join([self.lemmatizer.parse(word_token)[0].normal_form for word_token in
word_tokenize(text, language="russian")])

# Методы векторизации
class SequenceVectorizationMode(Enum):
    # Мешок слов
    BAG_OF_WORDS = 1
    # TF-IDF
    TF_IDF = 2
    # Word2Vec CBOW
    WORD2VEC_CBOW = 3
    # Word2Vec Skip gram
    WORD2VEC_SKIP_GRAM = 4

# Механизм векторизации
class SequenceVectorizationMechanism:
    # Инициализация объекта класса SequenceVectorizationMechanism
    # Входные параметры:
    # sequence_vectorization_mode (обязательный): метод векторизации - одно из значений
    SequenceVectorizationMode
    # params (необязательный): словарь с параметрами для метода векторизации
    def __init__(self, sequence_vectorization_mode, params={"num_words": 1000, "max_sequence_length":
1000, "window_size": 3, "vector_size": 100}):
        self.sequence_vectorization_mode = sequence_vectorization_mode
        self.params = params
        self.output_params = {}
        self.is_fitted = False
        if not isinstance(self.sequence_vectorization_mode, SequenceVectorizationMode):

```

```

        raise ValueError("Укажите в параметре sequence_vectorization_mode одно из значений
перечисления SequenceVectorizationMode")

# Обучение механизма векторизации
def fit(self, texts):
    self.tokenizer = None
    if self.sequence_vectorization_mode == SequenceVectorizationMode.BAG_OF_WORDS or
self.sequence_vectorization_mode == SequenceVectorizationMode.TF_IDF:
        self.tokenizer = self.fit_tokenizer(texts)
    elif self.sequence_vectorization_mode == SequenceVectorizationMode.WORD2VEC_CBOW:
        self.tokenizer = self.fit_word2vec(texts, False)
    elif self.sequence_vectorization_mode == SequenceVectorizationMode.WORD2VEC_SKIP_GRAM:
        self.tokenizer = self.fit_word2vec(texts, True)

    self.is_fitted = True
    return self.tokenizer

# Обучение токенайзера
def fit_tokenizer(self, texts):
    tokenizer = Tokenizer(num_words=self.params["num_words"], filters='', lower=False)
    tokenizer.fit_on_texts(texts)
    return tokenizer

# Обучение word2vec
def fit_word2vec(self, texts, is_skipgram):
    tokenized_texts = [word_tokenize(text, language="russian") for text in texts]
    word2vec = Word2Vec(sentences=tokenized_texts, vector_size=self.params["vector_size"],
window=self.params["window_size"], min_count=1, workers=4, sg=1 if is_skipgram else 0)
    vocabulary = word2vec.wv.key_to_index
    tokenizer = Tokenizer(num_words=self.params["num_words"], filters='', lower=False)
    tokenizer.word_index = vocabulary
    self.output_params["embedding_matrix"] = self.get_weight_matrix(word2vec)
    return tokenizer

# Возвращает матрицу весов
def get_weight_matrix(self, word2vec):
    num_words = self.params["num_words"]

    weight_matrix = np.zeros((num_words + 1, word2vec.vector_size))
    for i in range(num_words):
        weight_matrix[i + 1] = word2vec.wv[i]

    return weight_matrix

# Обработка набора текстов
def process(self, texts):
    if self.sequence_vectorization_mode == SequenceVectorizationMode.BAG_OF_WORDS:
        return self.tokenizer.texts_to_matrix(texts, mode='count')
    elif self.sequence_vectorization_mode == SequenceVectorizationMode.TF_IDF:
        return self.tokenizer.texts_to_matrix(texts, mode='tfidf')
    elif self.sequence_vectorization_mode == SequenceVectorizationMode.WORD2VEC_CBOW or
self.sequence_vectorization_mode == SequenceVectorizationMode.WORD2VEC_SKIP_GRAM:
        sequences = self.tokenizer.texts_to_sequences(texts)
        padded_sequences = pad_sequences(sequences, maxlen=self.params["max_sequence_length"])
        return padded_sequences
    else:
        return texts

# Методы балансирования текстового набора данных
class DatasetBalancingMode(Enum):
    # Даунсемплинг
    DOWNSAMPLING = 1
    # Апсемплинг
    UPSAMPLING = 2
    # Приведение количества примеров в каждом классе к среднему значению
    AVERAGING = 3

# Механизм балансирования текстового набора данных
class DatasetBalancingMechanism:
    # Инициализация объекта класса DatasetBalancingMechanism
    # Входные параметры:
    #   dataset_balancing_mode (обязательный): метод балансирования текстового набора данных - одно из
значений DatasetBalancingMode
    #   params (необязательный): словарь с параметрами для метода балансирования текстового набора
данных
    def __init__(self, dataset_balancing_mode, params={}):
        self.dataset_balancing_mode = dataset_balancing_mode
        self.params = params
        if not isinstance(self.dataset_balancing_mode, DatasetBalancingMode):

```

```

        raise ValueError("Укажите в параметре dataset_balancing_mode одно из значений перечисления DatasetBalancingMode")

# Обработка набора данных
def process(self, dataset, class_column_name):
    if self.dataset_balancing_mode == DatasetBalancingMode.DOWNSAMPLING:
        return self.downsampling(dataset, class_column_name, self.params)
    elif self.dataset_balancing_mode == DatasetBalancingMode.UPSAMPLING:
        return self.upsampling(dataset, class_column_name, self.params)
    elif self.dataset_balancing_mode == DatasetBalancingMode.AVERAGING:
        return self.averaging(dataset, class_column_name, self.params)
    else:
        return dataset

# Даунсемплинг
def downsampling(self, dataset, class_column_name, params):
    min_class_sample_count = dataset.groupby(class_column_name).count().min()[0]

    balanced_dataset = pd.DataFrame({})
    unique_column_values = dataset[class_column_name].unique()

    for unique_column_value in unique_column_values:
        unique_column_value_dataset = dataset[dataset[class_column_name] == unique_column_value]
        unique_column_value_dataset_columns = unique_column_value_dataset.columns
        unique_column_value_dataset_size = len(unique_column_value_dataset)

        if unique_column_value_dataset_size > min_class_sample_count:
            drop_indices = np.random.choice(unique_column_value_dataset.index,
            unique_column_value_dataset_size - min_class_sample_count, replace=False)
            unique_column_value_dataset = unique_column_value_dataset.drop(drop_indices)

        balanced_dataset = pd.concat([balanced_dataset, unique_column_value_dataset])

    return balanced_dataset

# Апсемплинг
def upsampling(self, dataset, class_column_name, params):
    max_class_sample_count = dataset.groupby(class_column_name).count().max()[0]

    balanced_dataset = pd.DataFrame({})
    unique_column_values = dataset[class_column_name].unique()

    for unique_column_value in unique_column_values:
        unique_column_value_dataset = dataset[dataset[class_column_name] == unique_column_value]
        unique_column_value_dataset_columns = unique_column_value_dataset.columns
        unique_column_value_dataset_size = len(unique_column_value_dataset)

        if unique_column_value_dataset_size < max_class_sample_count:
            row_repeat_count = int(max_class_sample_count / unique_column_value_dataset_size) + 1
            unique_column_value_dataset = pd.DataFrame(np.repeat(unique_column_value_dataset.values,
            row_repeat_count, axis=0))
            unique_column_value_dataset.columns = unique_column_value_dataset_columns
            unique_column_value_dataset_size = len(unique_column_value_dataset)

            drop_indices = np.random.choice(unique_column_value_dataset.index,
            unique_column_value_dataset_size - max_class_sample_count, replace=False)
            unique_column_value_dataset = unique_column_value_dataset.drop(drop_indices)

        balanced_dataset = pd.concat([balanced_dataset, unique_column_value_dataset])

    return balanced_dataset

# Приведение количества примеров в каждом классе к среднему значению
def averaging(self, dataset, class_column_name, params):
    total_sample_count = len(dataset)
    unique_column_values = dataset[class_column_name].unique()
    class_count = len(unique_column_values)
    average_sample_count = int(total_sample_count / class_count)

    balanced_dataset = pd.DataFrame({})

    for unique_column_value in unique_column_values:
        unique_column_value_dataset = dataset[dataset[class_column_name] == unique_column_value]
        unique_column_value_dataset_columns = unique_column_value_dataset.columns
        unique_column_value_dataset_size = len(unique_column_value_dataset)

        if unique_column_value_dataset_size > average_sample_count:
            drop_indices = np.random.choice(unique_column_value_dataset.index,
            unique_column_value_dataset_size - average_sample_count, replace=False)
            unique_column_value_dataset = unique_column_value_dataset.drop(drop_indices)

```

```

        elif unique_column_value_dataset_size < average_sample_count:
            row_repeat_count = int(average_sample_count / unique_column_value_dataset_size) + 1
            unique_column_value_dataset = pd.DataFrame(np.repeat(unique_column_value_dataset.values,
row_repeat_count, axis=0))
            unique_column_value_dataset.columns = unique_column_value_dataset.columns
            unique_column_value_dataset_size = len(unique_column_value_dataset)
            drop_indices = np.random.choice(unique_column_value_dataset.index,
unique_column_value_dataset_size - average_sample_count, replace=False)
            unique_column_value_dataset = unique_column_value_dataset.drop(drop_indices)

        balanced_dataset = pd.concat([balanced_dataset, unique_column_value_dataset])

    return balanced_dataset

# Препроцессор
class Preprocessor:
    # Инициализация объекта класса Preprocessor
    def __init__(self):
        self.simple_dataset_preprocessing_mechanisms = []
        self.simple_text_processing_mechanisms = []
        self.word_normalization_mechanism = None
        self.sequence_vectorization_mechanism =
SequenceVectorizationMechanism(SequenceVectorizationMode.BAG_OF_WORDS)
        self.dataset_balancing_mechanism = None
        self.output_params = {}
        self.is_preprocessed_train_dataset = False

    # Предобработка набора данных для обучения
    def preprocess_dataset_for_training(self, dataset, text_column_name="text",
class_column_name="class"):
        preprocessed_dataset = dataset.dropna()

        for simple_dataset_preprocessing_mechanism in self.simple_dataset_preprocessing_mechanisms:
            preprocessed_dataset = simple_dataset_preprocessing_mechanism.process(preprocessed_dataset)

        preprocessed_texts = preprocessed_dataset[text_column_name].tolist()
        for simple_text_processing_mechanism in self.simple_text_processing_mechanisms:
            preprocessed_texts = [simple_text_processing_mechanism.process(preprocessed_text) for
preprocessed_text in preprocessed_texts]

        if self.word_normalization_mechanism is not None:
            preprocessed_texts = [self.word_normalization_mechanism.process(preprocessed_text) for
preprocessed_text in preprocessed_texts]

        print(preprocessed_texts[0])

        self.output_params["tokenizer"] = self.sequence_vectorization_mechanism.fit(preprocessed_texts)
        self.output_params["embedding_matrix"] =
self.sequence_vectorization_mechanism.output_params.get("embedding_matrix")
        vectorized_texts = self.sequence_vectorization_mechanism.process(preprocessed_texts)

        self.append_to_data_frame(preprocessed_dataset, vectorized_texts, 'preprocessed_texts')

        if self.dataset_balancing_mechanism is not None:
            preprocessed_dataset = self.dataset_balancing_mechanism.process(preprocessed_dataset,
class_column_name)

        self.is_preprocessed_train_dataset = True

        return preprocessed_dataset

    # Предобработка набора данных для эксплуатации
    def preprocess_dataset_for_evaluating(self, dataset, text_column_name="text",
class_column_name="class"):
        if not self.is_preprocessed_train_dataset:
            raise ValueError("Сначала обучите препроцессор на наборе данных для обучения")

        preprocessed_dataset = dataset.dropna()

        for simple_dataset_preprocessing_mechanism in self.simple_dataset_preprocessing_mechanisms:
            preprocessed_dataset = simple_dataset_preprocessing_mechanism.process(preprocessed_dataset)

        preprocessed_texts = preprocessed_dataset[text_column_name].tolist()
        for simple_text_processing_mechanism in self.simple_text_processing_mechanisms:
            preprocessed_texts = [simple_text_processing_mechanism.process(preprocessed_text) for
preprocessed_text in preprocessed_texts]

        if self.word_normalization_mechanism is not None:
            preprocessed_texts = [self.word_normalization_mechanism.process(preprocessed_text) for
preprocessed_text in preprocessed_texts]

```

```

        vectorized_texts = self.sequence_vectorization_mechanism.process(preprocessed_texts)

        self.append_to_data_frame(preprocessed_dataset, vectorized_texts, 'preprocessed_texts')

        if self.dataset_balancing_mechanism is not None:
            preprocessed_dataset = self.dataset_balancing_mechanism.process(preprocessed_dataset)

        return preprocessed_dataset

# Добавление предобработки
def add_preprocessing(self, preprocessing_mechanism):
    if isinstance(preprocessing_mechanism, SimpleDatasetPreprocessingMechanism):
        self.add_if_not_contains(self.simple_dataset_preprocessing_mechanisms,
                                preprocessing_mechanism)
    elif isinstance(preprocessing_mechanism, SimpleTextPreprocessingMechanism):
        self.add_if_not_contains(self.simple_text_processing_mechanisms, preprocessing_mechanism)
    elif isinstance(preprocessing_mechanism, WordNormalizationMechanism):
        self.word_normalization_mechanism = preprocessing_mechanism
    elif isinstance(preprocessing_mechanism, SequenceVectorizationMechanism):
        self.sequence_vectorization_mechanism = preprocessing_mechanism
    elif isinstance(preprocessing_mechanism, DatasetBalancingMechanism):
        self.dataset_balancing_mechanism = preprocessing_mechanism
    else:
        raise ValueError("Неизвестный тип механизма предобработки")

# Добавление списка к набору данных
def append_to_data_frame(self, data_frame, tensors, column_name):
    tensor_list = self.tensor_to_list(tensors)
    data_frame[column_name] = pd.Series(tensor_list, index=data_frame.index)

# Преобразование тензоров в список
def tensor_to_list(self, tensors):
    tensor_list = []
    for tensor in tensors:
        tensor_list.append(tensor)

    return tensor_list

# Добавляет в коллекцию, только если элемента в ней еще нет
def add_if_not_contains(self, collection, item):
    if item not in collection:
        collection.append(item)

```

Значимые методы приложения для обучения моделей (training_module.ipynb)

```

# Загрузка CSV-файла
def load_csv(path):
    return pd.read_csv(path, delimiter=";")

# Проверка корректности набора данных
def is_dataset_valid(dataset):
    does_first_column_have_valid_name =
    does_second_column_have_valid_name =

    if not (does_first_column_have_valid_name and does_second_column_have_valid_name):
        raise ValueError("Набор данных не соответствует требованиям")

# Разбиение набора данных на обучающую и тестовую выборку
def split_dataset(dataset, test_dataset_percent):
    if test_dataset_percent >= 1.0:
        raise ValueError("Процент для тестового набора данных должен быть <= 1.0")

# Создадим DataFrame для обучающей/тестовой выборки
df_columns = {'text': pd.Series(dtype='str'),
              'class': pd.Series(dtype='str')}
train_df = pd.DataFrame(df_columns)
test_df = pd.DataFrame(df_columns)
# Сгруппируем набор данных по классам
class_groups = dataset.groupby(['class'])
# Заполним DataFrame для обучающей/тестовой выборки
for class_group in class_groups:
    dataset = class_group[1]
    total_count = len(dataset.index)
    train_dataset_percentage = 1 - test_dataset_percent
    train_dataset_count = int(total_count * train_dataset_percentage)

```

```

train_dataset = dataset.head(train_dataset_count)
test_dataset = dataset.copy().drop(train_dataset.index)

train_df = pd.concat([train_df, train_dataset])
test_df = pd.concat([test_df, test_dataset])

# Перемешиваем данные в выборках
train_df = train_df.sample(frac=1)
test_df = test_df.sample(frac=1)

return train_df, test_df

# Вычисление специфичности
def specificity(y_true, y_pred):
    neg_y_true = 1 - y_true
    neg_y_pred = 1 - y_pred
    fp = K.sum(neg_y_true * y_pred)
    tn = K.sum(neg_y_true * neg_y_pred)
    specificity = tn / (tn + fp + K.epsilon())
    return specificity

# Возвращает скомпилированную модель
def get_model(preprocessor, tensors, labels):
    model = Sequential()
    model.add(Embedding(2001, 100, mask_zero=False,
weights=[preprocessor.output_params['embedding_matrix']], input_length=tensors.shape[1],
trainable=False))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(100, dropout=0.2))
    model.add(Dense(labels.shape[1], activation='softmax'))
    model.compile(optimizer='adam',
loss='categorical_crossentropy',
metrics=['accuracy', tf.keras.metrics.Recall(name="sensitivity"), specificity])

return model

# Обучение модели
def train_model(data_frame, preprocessor, model, epochs, batch_size):
    # Получим метки
    labels = pd.get_dummies(preprocessed_dataset['section']).values
    print(model.summary())

    arrs = data_frame["preprocessed_texts"].tolist()
    sequences_np = np.asarray(arrs)

    history = model.fit(sequences_np, labels, epochs=epochs,
batch_size=batch_size, validation_split=0.1, callbacks=[EarlyStopping(monitor='val_loss', patience=3,
min_delta=0.0001, restore_best_weights=True)])

    return model, history

# Оценка качества модели
def evaluate_model(test_data_frame, model, history):
    arrs = test_preprocessed_dataset["preprocessed_texts"].tolist()
    test_sequences_np = np.asarray(arrs)
    test_labels = pd.get_dummies(test_preprocessed_dataset['section']).values
    # Оценим качество обучения
    accr = model.evaluate(test_sequences_np, test_labels)
    print("Оценка качества модели:")
    plot(history, 'accuracy', 'Точность')
    plot(history, 'sensitivity', 'Чувствительность')
    plot(history, 'specificity', 'Специфичность')
    print("Обучающая выборка")
    print("-----")
    print("Значение потерь на обучающей выборке: %f" % (min(history.history['loss'])))
    print("Точность на обучающей выборке: %f" % (min(history.history['accuracy'])))
    print("Чувствительность на обучающей выборке: %f" % (min(history.history['sensitivity'])))
    print("Специфичность на обучающей выборке: %f" % (min(history.history['specificity'])))
    print("Валидационная выборка")
    print("-----")
    print("Значение потерь на обучающей выборке: %f" % (min(history.history['val_loss'])))
    print("Точность на обучающей выборке: %f" % (min(history.history['val_accuracy'])))
    print("Чувствительность на обучающей выборке: %f" % (min(history.history['val_sensitivity'])))
    print("Специфичность на обучающей выборке: %f" % (min(history.history['val_specificity'])))
    print("Тестовая выборка")
    print("-----")
    print("Значение потерь на тестовой выборке: %f" % (accr[0]))
    print("Точность на тестовой выборке: %f" % (accr[1]))
    print("Чувствительность на тестовой выборке: %f" % (accr[2]))
    print("Специфичность на тестовой выборке: %f" % (accr[3]))

```

```
# Построение графика
def plot(history, metric, plot_name):
    plt.plot(history.history[metric])
    plt.plot(history.history['val_' + metric])
    plt.title(plot_name)
    plt.ylabel(metric)
    plt.xlabel('Эпоха')
    plt.legend(['Обучающая', 'Валидационная'], loc='upper left')
    plt.show()
```