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.

«10» июня 2021 г.

**ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ**

**РАБОТА**

**(бакалавриат)**

на тему

**«Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки»**

Направление подготовки: 09.03.04 Программная инженерия

Наименование ОПОП: Программная инженерия

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Цуканова Н.И.)

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (Головкин Н.В.)

Рязань 2021

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.

«11» мая 2021 г.

**ЗАДАНИЕ**

на выпускную квалификационную работу

Обучающемуся Головкину Никите Вячеславовичу, студенту группы 743 \_\_\_

(фамилия, имя, отчество, № группы)

1. Тема ВКР Разработка программного обеспечения для диагностики\_\_\_\_\_\_\_\_\_\_\_\_\_\_ заболеваний дыхательной системы человека по рентгенограмме грудной клетки

2. Срок представления законченной ВКР к защите: «10» июня 2021 г. \_

3. Руководитель Цуканова Нина Ивановна, РГРТУ, доцент \_

(фамилия, имя, отчество полностью, место работы, должность)

4. Исходные данные к ВКР Keras 2.4.0 \_\_\_

Python 3.8 \_\_

Java SE Development Kit 8u202 \_\_\_\_

Spring Framework 5.3.7 \_\_

Deep Java Library 0.11.0 \_\_

Ubuntu 20.04 \_\_

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: «11» мая 2021 г.

Руководитель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

Задание принял к исполнению «11» мая 2021 г.

Обучающийся \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись)

**АННОТАЦИЯ**

Пояснительная записка – 127 страниц, 7 таблиц, 56 рисунков, 34 источника.

Объект исследования – диагностика заболеваний дыхательной системы человека по рентгенограмме грудной клетки.

Цель выпускной квалификационной работы – «Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки».

Средства разработки – интегрированные среды разработки JetBrains IntelliJ IDEA 2020.1 и JetBrains PyCharm 2020.1, языки программирования Python и Java, библиотека глубокого обучения Keras и фреймворк Spring Framework.

Назначение разрабатываемого программного обеспечения – повышение точности диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки.

Область применения – рентгенологическая диагностика заболеваний.

Результат работы – проведен анализ предметной области, рассмотрены аналоги, спроектировано, разработано и протестировано программное обеспечение, составлена программная документация.

Значимость работы – разработанное программное обеспечение позволяет повысить точность диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки, снижает когнитивную нагрузку на рентгенолога и дает ему возможность сосредоточиться на тех рентгенограммах, где патологии присутствуют с большей вероятностью.

**ABSTRACT**

The explanatory note consists of 127 pages, 7 tables, 56 figures and 34 sources.

The object of the study is the diagnosis of human respiratory system diseases on chest X-ray.

The aim of the graduate qualification work is the development of software for the diagnosis of human respiratory system diseases on chest X-ray.

Development tools – integrated development environments JetBrains IntelliJ IDEA 2020.1 and JetBrains PyCharm 2020.1, programming languages Python and Java, deep learning library Keras and Spring Framework.

The purpose of the developed software is to increase the accuracy of human respiratory system disease diagnostics on chest X-ray.

Scope: radiological diagnosis of diseases.

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

The significance of the work is that the developed software allows to increase the accuracy of diagnosis of human respiratory system diseases on chest X-ray, reduces the cognitive load on a radiologist and allows him to focus on those radiographs where the pathology is more likely to be present.

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 8](#_Toc136779365)

[1 ПОСТАНОВКА ЗАДАЧИ 11](#_Toc136779366)

[2 ТЕХНИКО-ЭКОНОМИЧЕСКОЕ ОБОСНОВАНИЕ 13](#_Toc136779367)

[3 ОБОСНОВАНИЕ ВЫБОРА СРЕДСТВ РАЗРАБОТКИ 15](#_Toc136779368)

[4 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 17](#_Toc136779369)

[4.1 Основные сведения о машинном обучении 17](#_Toc136779370)

[4.2 Введение в искусственные нейронные сети 18](#_Toc136779371)

[4.3 Предобработка текстового набора данных 23](#_Toc136779372)

[4.3.1 Простые методы предобработки 23](#_Toc136779373)

[4.3.2 Методы нормализации 26](#_Toc136779374)

[4.3.3 Методы векторизации 28](#_Toc136779375)

[4.3.4 Методы балансирования 34](#_Toc136779376)

[4.4 Архитектуры нейронных сетей для классификации текста 36](#_Toc136779377)

[4.4.1 Рекуррентные нейронные сети 36](#_Toc136779378)

[4.4.2 Нейронные сети с долгой краткосрочной памятью 37](#_Toc136779379)

[4.5 Решение задачи классификации 37](#_Toc136779380)

[4.6 Оценка качества обучения 38](#_Toc136779381)

[5 ПРАКТИЧЕСКАЯ ЧАСТЬ 42](#_Toc136779382)

[5.1 Проектирование функциональности программного обеспечения 42](#_Toc136779383)

[5.2 Разработка архитектуры программного обеспечения 44](#_Toc136779384)

[5.3 Разработка модуля предобработки текстового набора данных 45](#_Toc136779385)

[5.3.1 Описание входных и выходных данных 46](#_Toc136779386)

[5.3.2 Разработка алгоритмов 46](#_Toc136779387)

[5.3.3 Программная реализация 48](#_Toc136779388)

[5.4 Разработка приложения для обучения моделей 52](#_Toc136779389)

[5.4.1 Описание входных и выходных данных 53](#_Toc136779390)

[5.4.2 Разработка алгоритмов 54](#_Toc136779391)

[5.4.3 Программная реализация алгоритмов 54](#_Toc136779392)

[5.5 Разработка веб-приложения 55](#_Toc136779393)

[5.5.1 Разработка алгоритма 56](#_Toc136779394)

[5.5.2 Разработка классов 56](#_Toc136779395)

[5.5.3 Программная реализация 57](#_Toc136779396)

[5.5.4 Разработка пользовательского интерфейса 58](#_Toc136779397)

[6 ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ 60](#_Toc136779398)

[6.1 Описание применения 60](#_Toc136779399)

[6.1.1 Назначение программы 60](#_Toc136779400)

[6.1.2 Условия применения 60](#_Toc136779401)

[6.1.3 Описание задачи 62](#_Toc136779402)

[6.1.4 Входные и выходные данные 62](#_Toc136779403)

[6.2 Руководство оператора 64](#_Toc136779404)

[6.2.1 Назначение программы 64](#_Toc136779405)

[6.2.2 Условия выполнения программы 64](#_Toc136779406)

[6.2.3 Выполнение программы 65](#_Toc136779407)

[6.2.4 Сообщения оператору 68](#_Toc136779408)

[6.3 Руководство программиста 69](#_Toc136779409)

[6.3.1 Назначение и условия применения программы 69](#_Toc136779410)

[6.3.2 Обращения к программе 71](#_Toc136779411)

[6.3.3 Входные и выходные данные 71](#_Toc136779412)

[6.3.4 Сообщения 71](#_Toc136779413)

[6.4 Руководство системного программиста 71](#_Toc136779414)

[6.4.1 Общие сведения о программе 71](#_Toc136779415)

[6.4.2 Структура программы 73](#_Toc136779416)

[6.4.3 Настройка программы 73](#_Toc136779417)

[6.4.4 Проверка программы 73](#_Toc136779418)

[6.4.5 Сообщения системному программисту 73](#_Toc136779419)

[7 ТЕСТИРОВАНИЕ 75](#_Toc136779420)

[7.1 Приложение для обучения моделей 75](#_Toc136779421)

[7.1.1 План тестирования 75](#_Toc136779422)

[7.1.2 Результаты тестирования 75](#_Toc136779423)

[7.1.3 Анализ результатов тестирования 76](#_Toc136779424)

[7.1.4 Исследование влияния предобработки текста на точность классификации текстовых данных 76](#_Toc136779425)

[7.2 Веб-приложение 76](#_Toc136779426)

[7.2.1 План тестирования 76](#_Toc136779427)

[7.2.2 Результаты тестирования 77](#_Toc136779428)

[7.2.3 Анализ результатов тестирования 78](#_Toc136779429)

[ЗАКЛЮЧЕНИЕ 79](#_Toc136779430)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 81](#_Toc136779431)

[ПРИЛОЖЕНИЕ А. ЛИСТИНГ НАИБОЛЕЕ ЗНАЧИМЫХ ЧАСТЕЙ ПРОГРАММЫ 84](#_Toc136779432)

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.

# 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.

# 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.

# 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.

# ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

## 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 6).

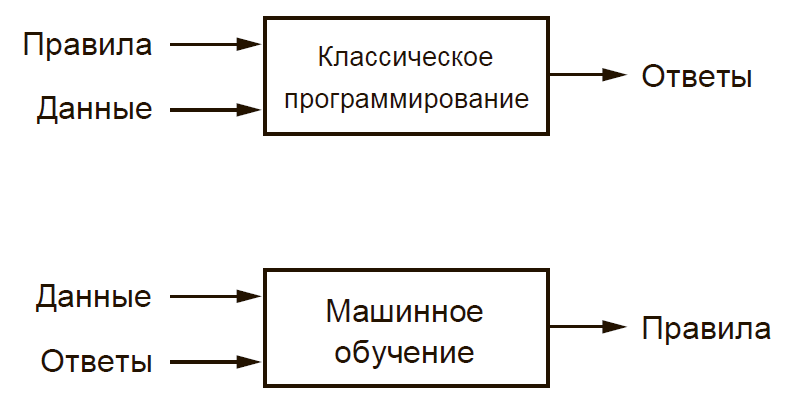


Рисунок 6 – Различия между классическим программированием и машинным обучение

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.

## 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 7 приведено схематическое представление полносвязной сети – сети, в которой каждый нейрон предыдущего слоя связан с каждым нейронном следующего.

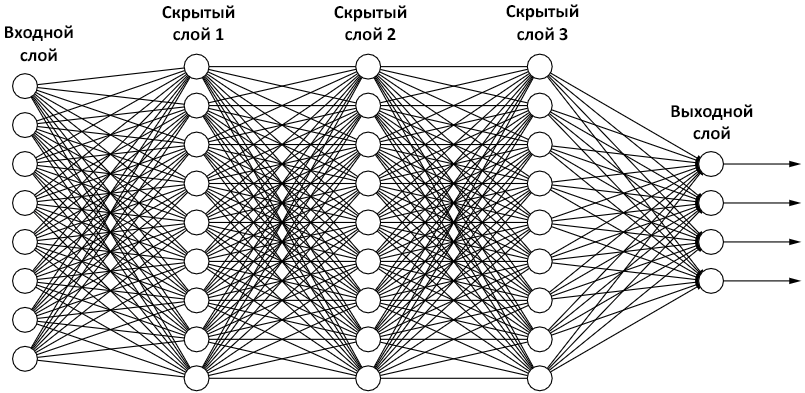


Рисунок 7 – Схематическое представление полносвязной сети

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 видеопамяти 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 (2) для .

где – градиент функции , – функция нейронной сети, – весовые коэффициенты нейронной сети.

Это полиномиальное уравнение из переменных, где – количество параметров в сети. Хотя такое уравнение можно решить для или , 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 и соответствующих меток .
2. Выполнить прямой ход на взятом батче и получить предсказания .
3. Вычислить значение потерь сети на данном батче между и .
4. Вычислить градиент значения потерь относительно параметров сети (обратный ход).
5. Немного изменить параметры в направлении, противоположному 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 8 приведена визуальная интерпретация 3D тензора.

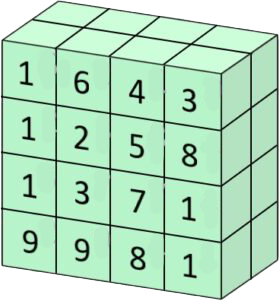


Рисунок 8 – Визуальная интерпретация 3D тензора

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.

## 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.

### 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 случае 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.

### 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.

### 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 будет 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 3, 4, 5):

где – это определенное слово, – это число вхождений слова в документ, – количество документов, – количество слов в документе .

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

где – это частота слова, – 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 |

Продолжение таблицы 3

|  |  |  |  |
| --- | --- | --- | --- |
| 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 |

Продолжение таблицы 5

|  |  |  |  |
| --- | --- | --- | --- |
| 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.

### Методы балансирования

При решении задачи классификации необходимо учитывать проблему несбалансированности классов.

Несбалансированность классов – это проблема, при которой на один пример в одном классе приходятся 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 |
| 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 |

Продолжение таблицы 8

|  |  |
| --- | --- |
| Адаптивные технологии в современных автоматизированных обучающих системах | 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 |

**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.

## 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.

### 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.

### 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.

## 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 классифкация 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.

## 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 6)

где — номер итерации обучения, – количество отработанных моделью примеров, – желаемый выход модели, – 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 22 приведено проявление феномена переобучения на графике значений потерь.

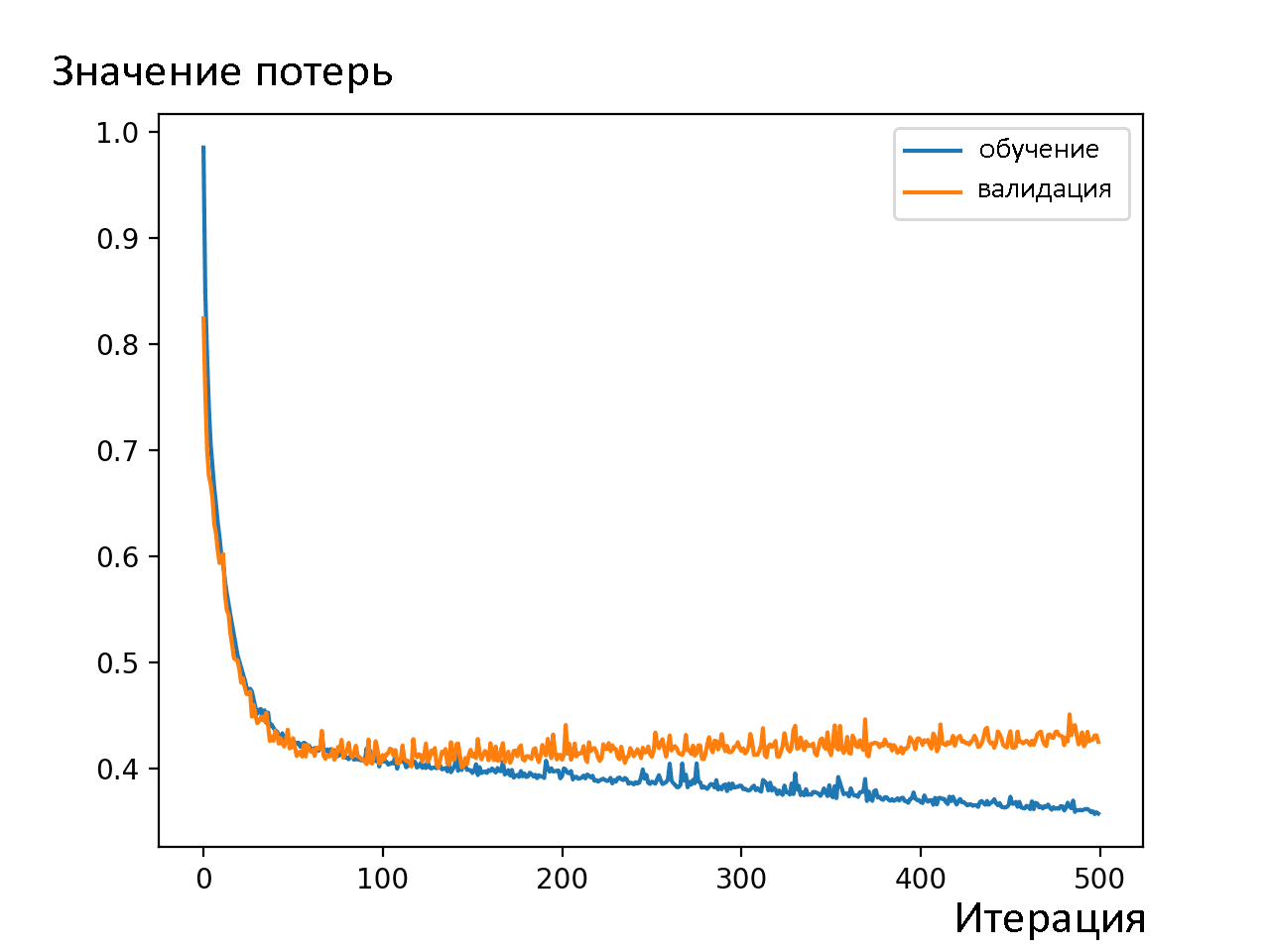


Рисунок 22 – Проявление феномена переобучения на графике значений потерь

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 7).

где LOSS – значение потерь, n – количество примеров, – вероятность i-го примера, – 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 8)

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 9)

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 10)

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.

# ПРАКТИЧЕСКАЯ ЧАСТЬ

## 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 24 приведено представление вариантов использования.

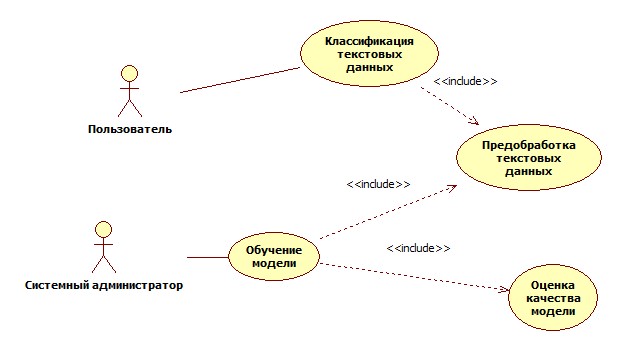


Рисунок 24 – Представление вариантов использования

На рисунке 24 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.  (a) 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 |

## 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.

## 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 использование языка программирования Python и библиотек NLTK, PyMorphy2 и Gensim. В качестве среды разработки используется IntelliJ PyCharm 2020.1 Community Edtion.

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

### 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.

### 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 … приведен алгоритм работы класса Preprocesor при предобработке текстового набора данных для обучения.

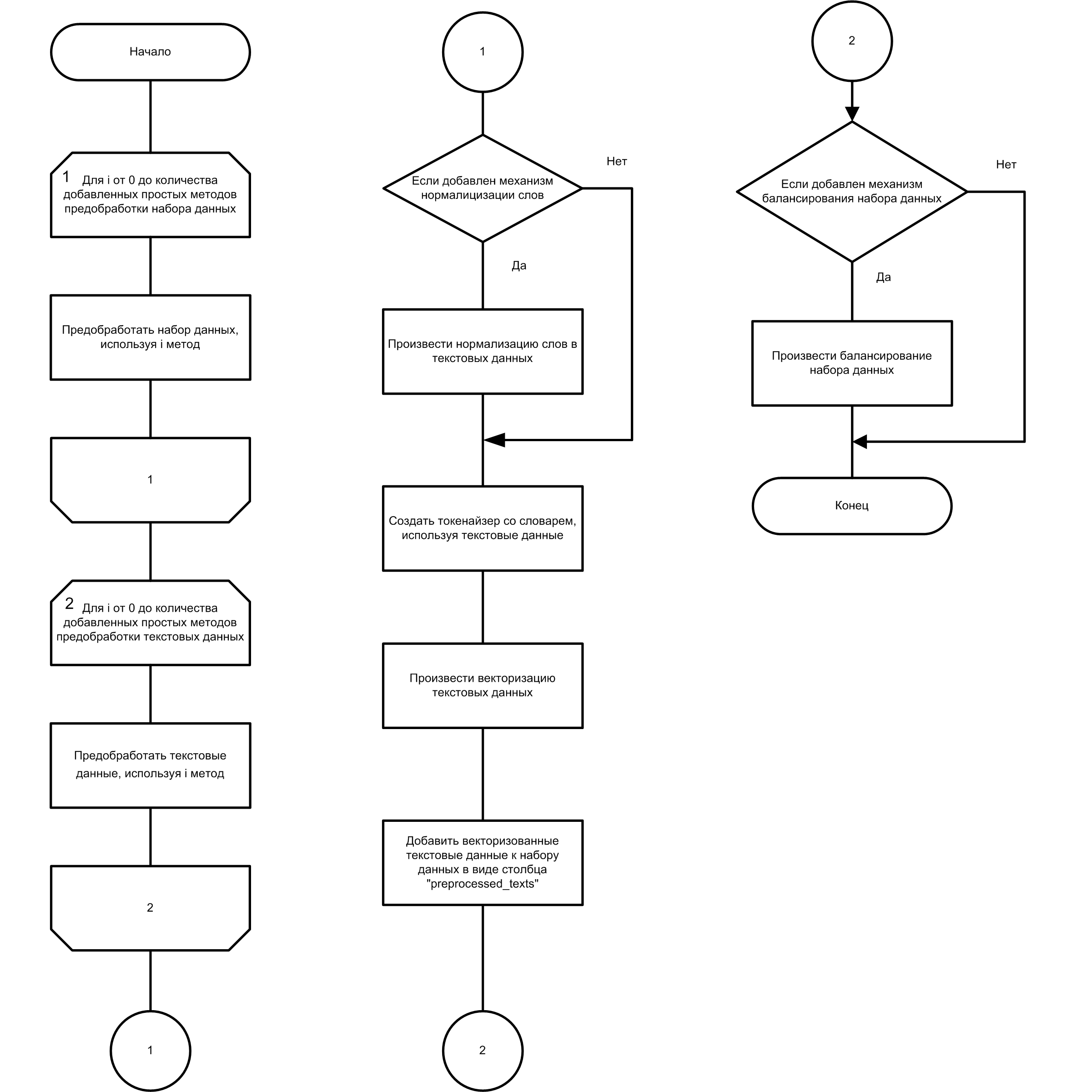


Рисунок 24 – Алгоритм работы класса Preprocessor при предобработке текстовго набора данных для обучения

### 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 пунктации 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 PyMorphy2.

**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 коилчество 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.

## 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.

### 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 26 изображен фрагмент такого файла.

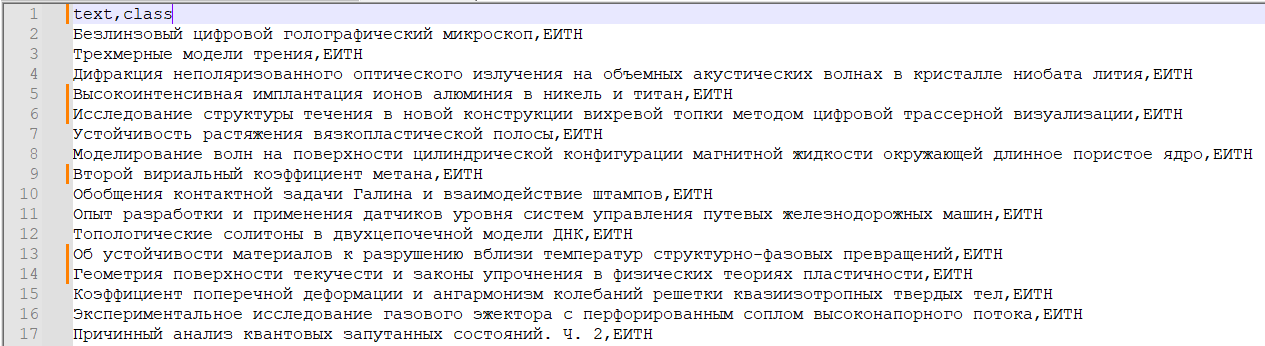


Рисунок 26 – Фрагмент CSV-файла

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.

### Разработка алгоритмов

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

На рисунке 27 приведен разработанный алгоритм.

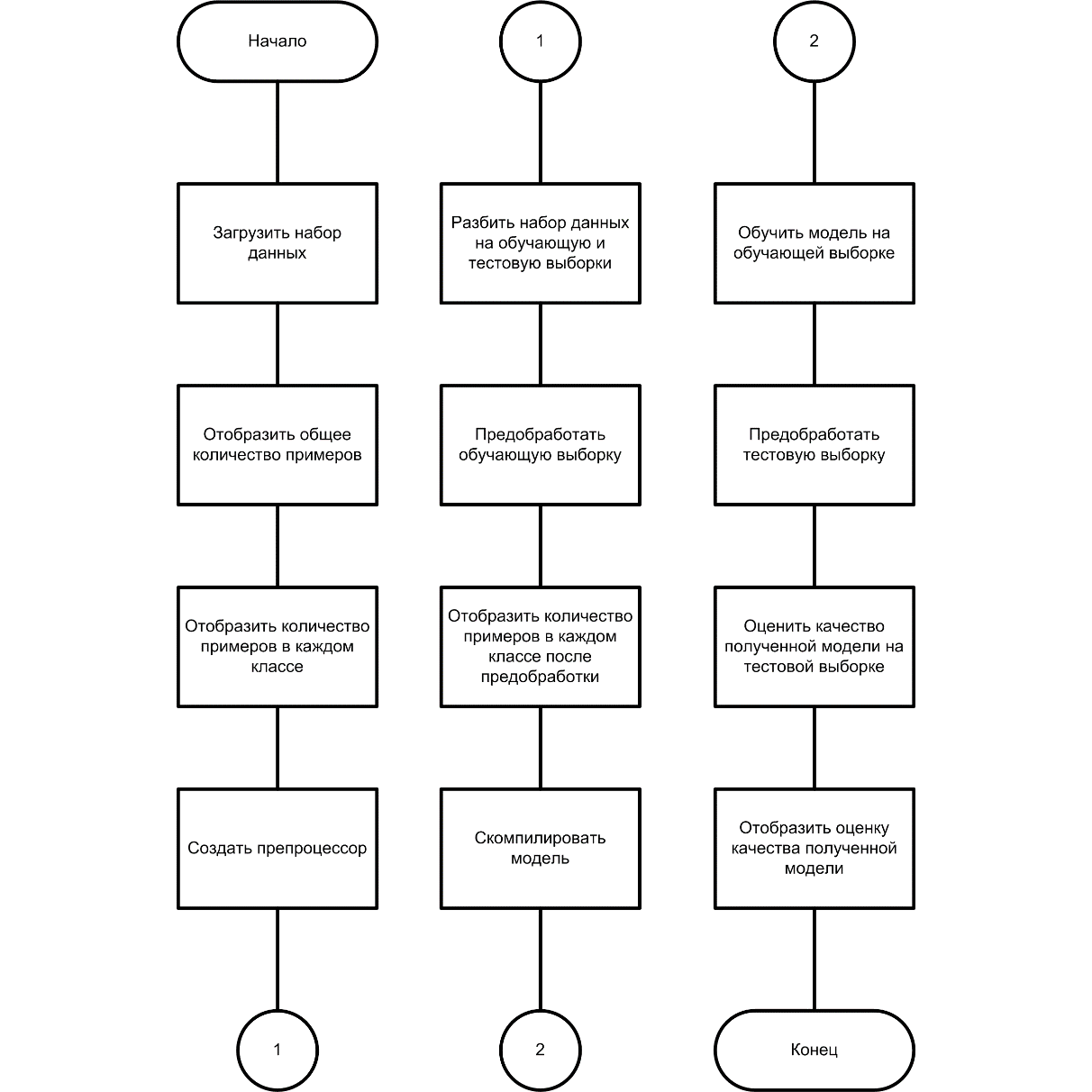


Рисунок 27 – Общий алгоритм работы приложения для обучения моделей

Перейдем к реализации разработанного алгоритма.

### 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 CSV-файла на соответствие требованиям**

Функция is\_csv\_valid(data\_frame) позволяет проверить соответствие загруженного CSV-файла требованиям, предъявленным в 5.4.1. Входной параметр: объект класса data\_frame. Выход: True или False, в зависимости от того, прошел ли CSV-файл проверку.

**Разбиение данных на 2 выборки**

Функция split\_dataset(data\_frame, test\_set\_percent) производит разбиение таблицы набора данных data\_frame на 2 выборки – обучающую и тестовую. Входные параметры: data\_frame – таблица с набором данных, test\_set\_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, text\_column\_name, class\_column\_name) позволяет обучить модель. Входные параметры: data\_frame – таблица с данными, preprocessor – препроцессор, model – модель, которую необходимо обучить, text\_column\_name – название столбца с текстами, class\_column\_name – название столбца с классами. Выходные значения – обученная модель и история обучения.

**Оценка качества модели**

Метод evaluate\_model(data\_frame, preprocessor, model, text\_column\_name, class\_column\_name) производит оценку качества модели, применяя метрики, рассмотренные в 4.9. Выходное значение – объект класса ModelEvaluationResult.

**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.

## 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 38 схематично представлено взаимодействие между элементами шаблона.



Рисунок 38 – Взаимодействие между элементами шаблона MVC

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

### 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 39 приведены разработанные алгоритмы.

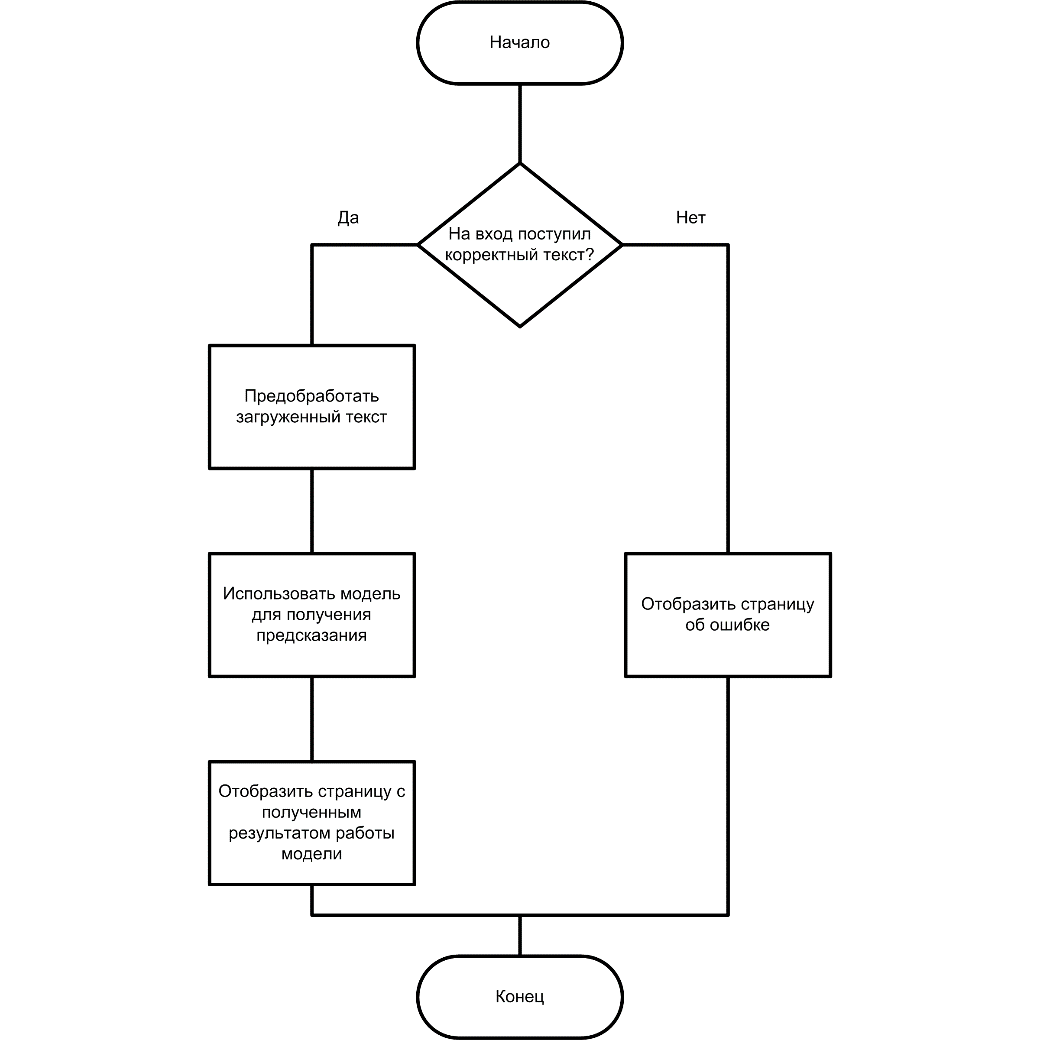


Рисунок 39 – Алгоритм классификации по поступающему на вход тексту

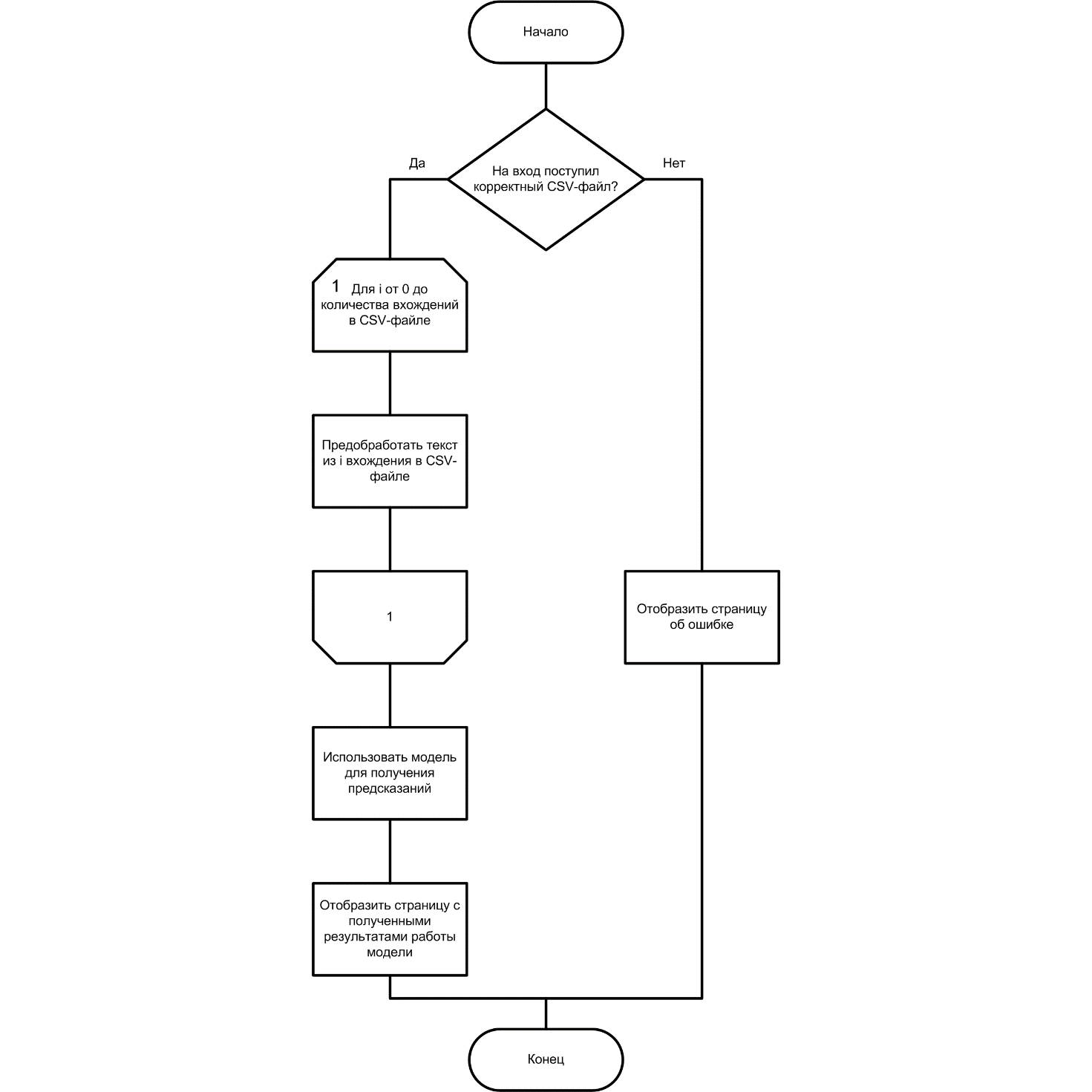


Рисунок 39 – Алгоритм классификации по поступающему на вход CSV-файлу

### Разработка классов

Для веб-приложения были созданы классы, описание которых приведено в таблице 6.

Таблица 6 – 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 |

### 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.

### 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 41 приведен вид панели навигации.

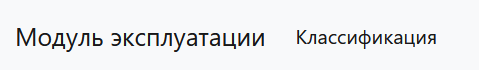


Рисунок 41 – Панель навигации

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



Рисунок 42 – Главная страница

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



Рисунок 47 – 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 … приведен вид этой страницы.



Рисунок 47 – Страница с результатами классификации по CSV-файлу

Страница, выводящая сообщения о возникающих ошибках должна содержать сообщение о произошедшей ошибке. Ниже на рисунке 45 приведен вид данной страницы.

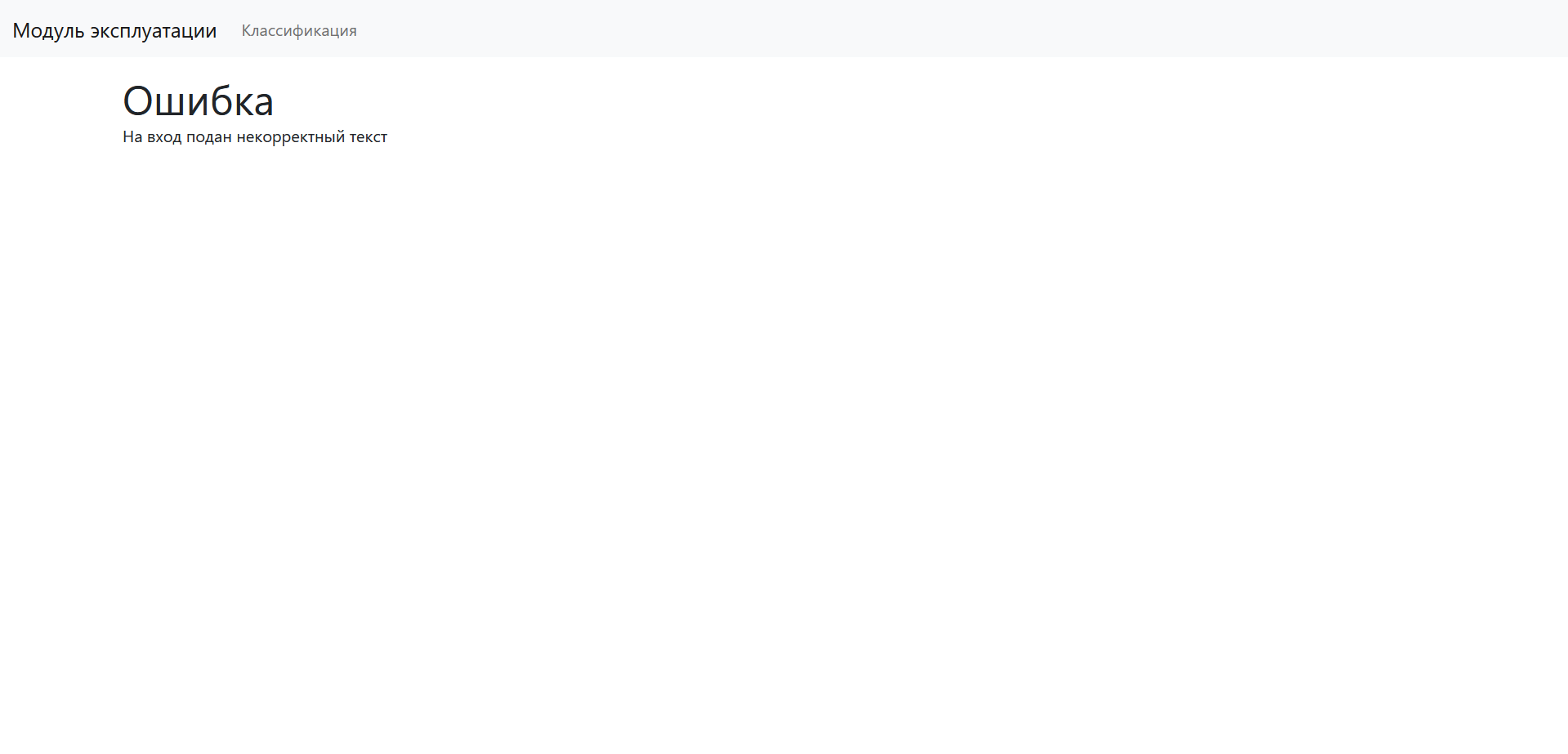


Рисунок 45 – Страница, выводящая сообщения о возникающих ошибках

# ПРОГРАММНАЯ ДОКУМЕНТАЦИЯ

## Описание применения

### 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.

### 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.

### 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.

### 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.

## Руководство оператора

### 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.

### 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.

### 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 47 приведен результат выполнения данного шага.



Рисунок 47 – 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 ).

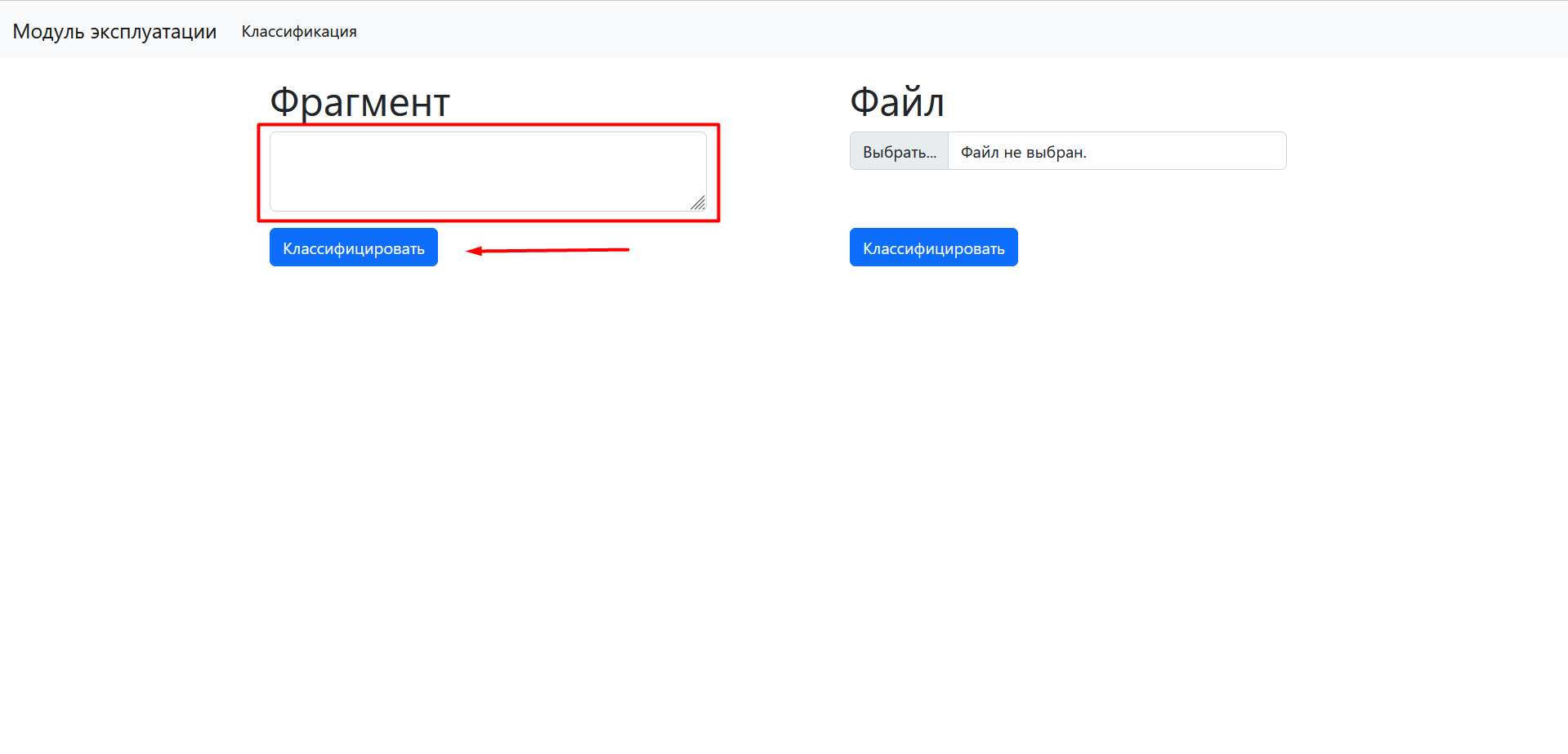


Рисунок 47 – Классификация по тексту

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



Рисунок 47 – 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 …)

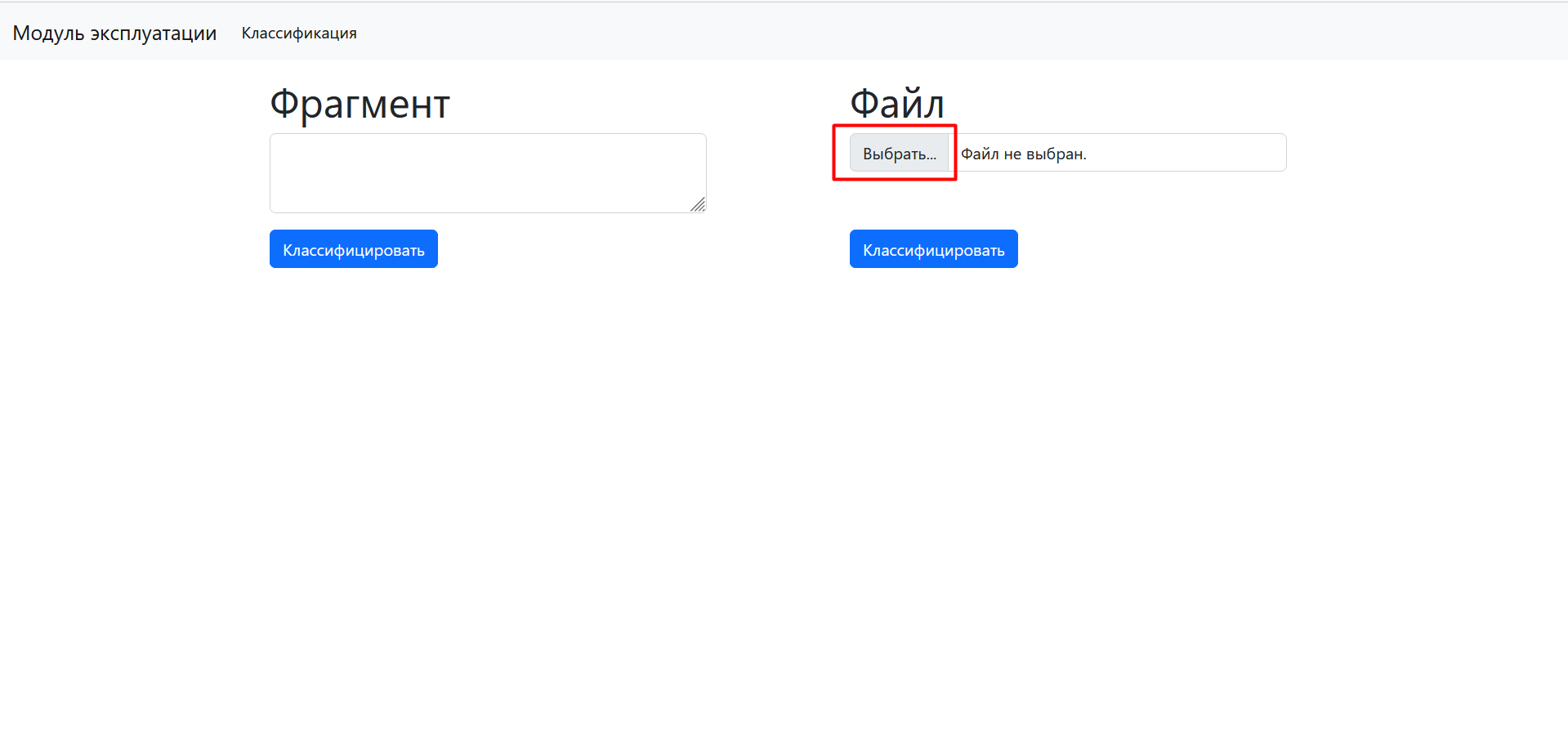


Рисунок 47 – Открытие диалога для выбора CSV-файла

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

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

\*рисунок\*

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 ).



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

\*рисунок\*

**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 ).

\*рисунок\*

Для обучения модели необходимо расположить набор данных в виде CSV-файла по пути … .

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 «,».

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

\*рисунок\*

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

\*рисунок\*

В результате обучения по пути … будет сохранена обученная модель, которую в дальнейшем можно использовать в веб-приложении (развертыванием модели занимается системный администратор).

Для исследования влияния предобработки текста на точность классификации текстовых данных необходимо запустить соответствующую ячейку (рисунок …).

\*рисунок\*

После ее выполнения в сводной таблице будут представлены оценки качества обученных моделей на текстовых данных, предобработанных различным способом (рисунок …).

\*рисунок\*

Соответствующие модели будут сохранены по пути … .

### 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».

## Руководство программиста

### 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 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 (https://github.com/googlecolab/colabtools/issues/2269)

### 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 7).

Таблица 7 – 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 |

### 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.

### Сообщения

Сообщения были описаны в 6.2.4.

## Руководство системного программиста

### 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 (https://github.com/googlecolab/colabtools/issues/2269)

### 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.

Сервер веб-приложения запускается при помощи интерпретатора языка Python (?).

### Настройка программы

Запуск веб-приложения на сервере производится следующим образом.

1. В директории с приложением необходимо разместить обученную модель trained\_model.keras.
2. Выполнить команду … для запуска веб-приложения.

### 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.

### 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.

# ТЕСТИРОВАНИЕ

## Исследование влияния предобработки текста на точность классификации текстовых данных

Для исследования влияния предобработки текста на точность классификации текстовых данных был выбран набор данных … .

Результаты исследования влияния предобработки текста приведены ниже в таблице … .

\*результат\*

## Приложение для обучения моделей

### 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.

### 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.

### 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.

## Веб-приложение

### 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 требования для классификации;
* приложение запускается, обученная модель найдена;
* приложение запускается, обученная модель не найдена.

### 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 требования 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.

### 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. Дарби М. и др. Клиническая интерпретация рентгенограммы легких. Справочник / пер. с англ. В.Н. Трояна. – М.: ГЭОТАР-Медиа, 2020. – 216 с.
2. Корн Дж., Пойнтон К. Рентгенография грудной клетки. / пер. с англ. В.А. Штабницкого. – М.: БИНОМ-Пресс, 2020. – 176 с.
3. Мартесен К.М. Рентгенология Техника исследований и анализ изображений. / пер. с англ. А.А. Пашкова. – М.: Издательство Панфилова, 2021. – 612 с.
4. Мельников В.В. Рентгенография в диагностике заболеваний органов грудной клетки: учебное пособие. – М.: ГЭОТАР-Медиа, 2017. – 190 с.
5. Монина Л.Н. Рентгенография. Качественный рентгенофазовый анализ. – М.: Проспект, 2021. – 120 с.
6. Фаулер М. Шаблоны корпоративных приложений. / пер. с. англ. Т.П. Кайгородовой. – М.: Вильямс, 2016. – 544 с.
7. Хофер М. Рентгенологическое исследование грудной клетки. Практическое руководство. / пер. с англ. В.В. Ипатова. – М.: Медицинская литература, 2008. – 224 с.
8. An introduction to Global Average Pooling in convolutional neural networks // Adventures in Machine Learning [сайт]. – URL: https://adventuresinmachinelearning.com/global-average-pooling-convolutional-neural-networks/ (дата обращения 03.06.2021).
9. Brederhoff J, Racoveanu N Radiological services throughout the world // Diagnostic Imaging. – 1982. – № 51. – pp. 121-133.
10. Chollet F. Deep Learning with Python. – Shelter Island: Manning Publications Co., 2018. – 384 p.
11. Core Technologies // Spring Framework Documentation [сайт]. – URL: https://docs.spring.io/spring-framework/docs/current/reference/html/core.html (дата обращения 06.06.2021).
12. Elliot T. The State of the Octoverse: machine learning // The GitHub Blog [сайт]. – URL: https://github.blog/2019-01-24-the-state-of-the-octoverse-machine-learning/ (дата обращения: 18.05.2021).
13. European Respiratory Society. The Global Impact of Respiratory Disease. — Sheffield: Forum of International Respiratory Societies (FIRS), 2017. — 42 p.
14. Fowler M. Inversion of Control Containers and the Dependency Injection pattern // Martin Fowler [сайт]. – URL: https://martinfowler.com/articles/injection.html (дата обращения 06.06.2021).
15. Geron A. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow. – Sebastopol: O'Reilly Media, 2019. – 856 p.
16. Goodfellow I. et al. Deep Learning. – Cambridge: MIT Press, 2016. – 800 p.
17. He K. et al. Deep Residual Learning for Image Recognition // 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) – 2016. – pp. 770-778.
18. Huang G. et al. Densely Connected Convolutional Networks // 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) – 2017. – pp. 2261-2269.
19. 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).
20. Ioffe S., Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift // Proceedings of Machine Learning Research. – 2015. – № 37 – pp. 448-456.
21. Irvin J. et al. CheXpert: A Large Chest Radiograph Dataset with Uncertainty Labels and Expert Comparison // Proceedings of the AAAI Conference on Artificial Intelligence. – 2019. – № 33. – pp. 590-597.
22. Johnson J., Khoshgoftaar T. Survey on deep learning with class imbalance // Journal of Big Data. – 2019. – № 6. – pp. 125-142.
23. Kermany D. et al. Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning // Cell. – 2018. – № 172. – pp. 1122-1131.
24. Krawczyk B. Learning from imbalanced data: open challenges and future directions // Progress in Artificial Intelligence. – 2016. – № 5, pp. 221–232.
25. Krishna M. et al. Image classification using Deep learning // International Journal of Engineering & Technology. – 2018. – № 614. – pp. 146-179.
26. Russakovsky O. et al. ImageNet Large Scale Visual Recognition Challenge // International Journal of Computer Vision. — 2014. — № 115. – pp. 789-862.
27. Scherer D. et al. Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition // International Conference on Artificial Neural Networks 2010. – 2010. – pp. 92-101.
28. Shorten C. A survey on Image Data Augmentation for Deep Learning. // Journal of Big Data. – 2019. – № 6. – pp. 345-389.
29. Simonyan K., Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition. // Journal of Deep Learning. – 2014. – № 16. – pp. 476-541.
30. Susmaga R. Confusion Matrix Visualization // Intelligent Information Processing and Web Mining. Advances in Soft Computing. – 2004. – № 25. – pp. 107-116.
31. 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).
32. 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).
33. Ying X. An Overview of Overfitting and its Solutions // Journal of Physics: Conference Series. – 2019. – № 1168 – pp. 314-356.
34. Zhuang F. et al. A Comprehensive Survey on Transfer Learning // Proceedings of the IEEE. – 2020. – pp. 1-34.

ПРИЛОЖЕНИЕ А. ЛИСТИНГ НАИБОЛЕЕ ЗНАЧИМЫХ ЧАСТЕЙ ПРОГРАММЫ

**Приложение для обучения моделей (train\_model.py)**

# Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

# Автор: Головкин Н.В., 09.03.04, 743

# Научный руководитель: Цуканова Н.И., доцент

# Средства разработки: Python 3.8, Keras 2.4.0

# Назначение данной части ПО: Приложение для обучения моделей

# Дата создания: 14.05.2021

import os

from keras.utils import ConfusionMatrixUtils

import keras

%matplotlib inline

import matplotlib.pyplot as plt

import matplotlib.image as mpimg

import tensorflow as tf

import pandas as pd

import numpy as np

from keras.preprocessing.image import load\_img

from keras.preprocessing.image import img\_to\_array

from keras.preprocessing.image import array\_to\_img

from keras.preprocessing import image

from keras.optimizers import Adam

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

from keras.preprocessing.image import ImageDataGenerator

from classification\_models.keras import Classifiers

from keras import backend

from keras import Sequential

from keras import Model

from keras.optimizers import Adam

# размер изображения, поступающего на вход модели

image\_size = 256

# позволяет получить в виде списка аргументы командной строки. Входные параметры отсутствуют. Выходное значение: список с аргументами командной строки

def get\_cmd\_args():

# аргументы командной строки

cmd\_args = sys.argv

if len(cmd\_args) == 1 or len(cmd\_args) == 2:

if os.path.isfile(args[0]):

return cmd\_args

else:

raise Exception("Указанный файл не существует")

else:

raise Exception("Неверное количество переданных аргументов")

# 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.

def load\_csv(path):

try:

# таблица с данными

data\_frame = pd.read\_csv(path)

print("Набор данных успешно загружен")

return data\_frame

except:

raise Exception("Не удалось загрузить набор данных")

# позволяет проверить соответствие загруженного CSV-файла требованиям, предъявленным в 5.3.1. Входной параметр: объект класса data\_frame. Выход: True или False, в зависимости от того, прошел ли CSV-файл проверку.

def is\_csv\_valid(data\_frame):

# первый столбец имеет название "Путь" (путь к файлу)

if not (data\_frame.columns[0] == "Путь"):

return False

# имеется не менее 1 столбца, кроме столбца "Путь"

if not len(data\_frame.columns) > 1:

return False

# столбец "Путь" в каждой клетке содержит путь к определенному изображению

file\_paths = data\_frame["Путь"]

for file\_path in file\_paths:

if not os.path.isfile(file\_path):

return False

# в каждой ячейке столбцов (кроме столбца "Путь") обязательно находится одно из 2 значений – "0" (непринадлежность к классу) и "1" (принадлежность к классу)

for column in data\_frame.columns:

# уникальные значения в столбце column

unique\_values = data\_frame[column].unique()

if !("0" in unique\_values and "2" in unique\_values and len(unique) == 2):

return False

# для каждого столбца (кроме столбца "Путь") обязательно должно присутствовать не менее 50 ячеек со значением "0" и не менее 50 ячеек со значение "1"

for column in range(1, len(data\_frame.columns)):

if not (data\_frame.loc[data\_frame[column] == '0'].count() >= 50 and data\_frame.loc[data\_frame[column] == '1'].count() >= 50):

return False

return True

# позволяет создать сбалансированный набор данных. Входные параметры: data\_frame – таблица с данными, classname – название класса, относительно которого будет производиться балансировка других классов. Выход: таблица, в которой данные сбалансированы по классам. Т.к. задача многозначной классификации в данном случае решается с использованием стратегии «один против всех», то в возвращаемой таблице всего 1 класс – «присутствие класса на изображении».

def create\_balanced\_dataset(data\_frame, classname):

# данные текущего класса

classname\_data = data\_frame[data\_frame[classname] == 1.0]

# все классы

classnames = data\_frame.columns.values

classnames.remove("Путь")

classnames.remove(classname)

# сколько нужно от каждого класса примеров, чтобы сделать набор сбалансированным

each\_classname\_needed\_sample\_count = int(len(classname\_data) / (len(classnames) + 1))

# от каждого класса нужное количество примеров

all\_other\_classnames\_data\_frame = data\_frame[data\_frame[classname\_data] == 0.0][:data\_frame.shape[0] - each\_classname\_needed\_sample\_count]

# определяем для каких строк отрицательные результаты

negative\_results = data\_frame[data\_frame[classnames[0]] == 0.0]

if len(classnames) > 1:

for i in range(1, len(classnames)):

negative\_results = negative\_results & data\_frame[data\_frame[classnames[i]] == 0.0]

# набор данных для нормальных рентгенов

no\_finding\_data\_frame = data\_frame.loc[negative\_results]

# нормальные рентгены + рентгены с болезнями кроме classname

all\_other\_classnames\_data\_frame = pd.concat([all\_other\_classnames\_data\_frame, no\_finding\_data[:each\_classname\_needed\_sample\_count]])

return pd.concat([our\_data\_frame, classname\_data])

# позволяет произвести предобработку данных. Возвращаемый им объект выполняет нормализацию и аугментацию данных во время обучения. Входной параметр: data\_frame – таблица с набором данных. Выход: объект класса ImageDataGenerator

def preprocess\_data(data\_frame):

return ImageDataGenerator(rescale=1.0/255.0, rotation\_range=20, width\_shift\_range=0.2, height\_shift\_range=0.2, horizontal\_flip=True)

# позволяет создать объект для подгрузки данных во время обучения с жесткого диска. Такой подход позволяет обойти ограничение, связанное с ограниченным размером оперативной памяти, которого часто бывает недостаточно для единовременной загрузки всех изображений из набора данных. Входные параметры: data\_frame – таблица с набором данных, image\_data\_generator – объект класса ImageDataGenerator, image\_size – размер (в виде целочисленного значения), к которому должны быть приведены все изображения. Выход: объект класса DataFrameIterator.

def сreate\_dataloader(data\_frame, image\_data\_generator, image\_size):

return image\_data\_generator.flow\_from\_dataframe(data\_frame, x\_col='Путь', y\_col='ПрисутствиеКласса', target\_size=image\_size, color\_mode='rgb', class\_mode='raw')

# производит разбиение таблицы набора данных data\_frame на 3 выборки – обучающую, валидационную и тестовую. Входные параметры: data\_frame – таблица с набором данных, validation\_set\_percent – процент примеров для валидационной выборки от общего числа примеров, test\_set\_percent – процент примеров для тестовой выборки от общего числа примеров. Выходные значения: 3 таблицы – для обучающей, валидационной и тестовой выборок. Если сумма validation\_set\_percent и test\_set\_percent >= 1, то выбрасывается исключение.

def split\_dataset(data\_frame, validation\_set\_percent, test\_set\_percent):

if validation\_set\_percent + test\_set\_percent >= 1:

raise Exception("Текст")

# тренировочный и валидационный наборы данных

train\_dataset, validation\_dataset

= train\_test\_split(train\_dataset, test\_size=validation\_set\_percent)

# тренировочный и тестовый наборы данных

train\_dataset, test\_dataset

= train\_test\_split(train\_dataset, test\_size=test\_set\_percent/(1-validation\_set\_percent))

return train\_dataset, validation\_dataset, test\_dataset;

# производит компиляцию модели. В этой функции описывается вся сверточная нейронной сети (ее архитектура, применяемые гиперпараметры, функция потерь и т.д.). Входной параметр – classname – название класса для которого компилируется модель. Выход: объект класса Model.

def compile\_model(classname):

# DenseNet121 и функция для предобработки для него

DenseNet121, preprocess\_input = Classifiers.get('densenet121')

# DenseNet121 для конкретных данных

base\_model = DenseNet121(input\_shape=(image\_size, image\_size, 3), weights='imagenet', include\_top=False)

# полносвязный слой для классификации

x = Dense(1, activation="sigmoid")(base\_model.output)

# готовая модель

model = Model(inputs=base\_model.inputs, outputs=x)

model.name = classname

# оптимизатор

opt = tf.keras.optimizers.Adam(lr=0.0002)

model.compile(optimizer=opt, loss='binary\_crossentropy', metrics=['accuracy', 'sensitivity', 'specificity'])

return model

# производит обучение модели до тех пор, пока не наступит переобучение. Входные параметры: model – объект класса Model (скомпилированная модель), train\_dataloader – объект подгрузки данных для обучающей выборки, validation\_dataloader – объект подгрузки данных для валидационной выборки. Выход: словарь, содержащий данные об обучении модели (значение потерь на каждой эпохе, точность на каждой эпохе и т.д.).

def train(model, train\_dataloader, validation\_dataloader):

try:

# заморозка весов

for i in range(1, len(model.layers - 1)):

model.layers[i].trainable = False

model.fit(train\_data\_flow, validation\_data=test\_data\_flow, epochs=10)

# разморозка весов

for i in range(1, len(model.layers - 1)):

model.layers[i].trainable = True

# обратный вызов, позволяющий остановить модель, если началось переобучение и вернуться к лучшим весовым коэффициентам

earlyStopping = EarlyStopping(monitor="val\_loss", patience=3, mode="min", restore\_best\_weights=True)

return model.fit(train\_data\_flow, validation\_data=test\_data\_flow, epochs=999999, callbacks=[earlyStopping])

except:

raise Exception("Во время обучения произошла ошибка")

# производит сохранение обученной модели по пути, где располагается само приложение. Входной параметр: model – обученная модель. Выходное значение отсутствует. Если при сохранении произошла ошибка, то выбрасывается исключение.

def save\_model(model):

try:

model.save('%s.h5' % (model.name))

except:

raise Exception("Не удалось сохранить модель")

# позволяет сохранить графики значения потерь и точности на обучающей и валидационной выборках. Сохранение производится по пути, где располагается само приложение. Входной параметр: model\_training\_history – словарь, содержащий данные об обучении модели, classname – название класса для которого была скомпилирована модель. Выходное значение отсутствует. Если при сохранении произошла ошибка, то выбрасывается исключение.

def save\_plots(model\_training\_history, classname):

try:

plt.plot(model\_training\_history.history['accuracy'])

plt.plot(model\_training\_history.history['val\_accuracy'])

plt.title('График точности')

plt.ylabel('Точность')

plt.xlabel('Эпоха')

plt.legend(['обучающая', 'валидационная'], loc='upper left')

plt.savefig("%s\_accuracy\_plot.png")

plt.clf()

plt.plot(model\_training\_history.history['loss'])

plt.plot(model\_training\_history.history['val\_loss'])

plt.title('График значения потерь')

plt.ylabel('Значение потерь')

plt.xlabel('Эпоха')

plt.legend(['обучающая', 'валидационная'], loc='upper left')

plt.savefig("%s\_loss\_plot.png")

except:

raise Exception("Не удалось сохранить графики")

plt.clf()

# производит тестирование модели на тестовой выборке, применяя метрики, рассмотренные в 4.9. Входные параметры: model – обученная модель, test\_dataloader - объект подгрузки данных для тестовой выборки. Выход: словарь с результатами тестирования модели.

def test\_model(model, test\_dataloader):

return model.evaluate(test\_dataloader)

# производит сохранение результатов тестирования (изображение матрицы ошибок, файл с результатами тестирования) по пути, где располагается само приложение. Входной параметр: test\_result – результаты тестирования модели. Выходное значение отсутствует. Если при сохранении произошла ошибка, то выбрасывается исключение

def save\_test\_results(test\_results, classname):

try:

with open('%s\_test\_results.txt' % (classname), 'w') as f:

print(test\_results, file=f)

ConfusionMatrixUtils.save\_confusion\_matrix(test\_results, classname, "%s\_confusion\_matrix.png" % (classname))

except:

raise Exception("Не удалось сохранить результаты тестирования")

# позволяет синтезировать модель. Входные параметры: data\_frame – таблица с данными, classname – название класса, относительно которого будет синтезироваться модель. Выходное значение отсутствует.

def synthesize\_model(data\_frame, classname):

# сбалансированный набор данных

balanced\_dataset = create\_balanced\_dataset(data\_frame, classname)

# объект, предобрабатывающий изображения на входе

image\_data\_generator = preprocess\_data(data\_frame)

# массив с выборками

datasets = split\_dataset(data\_frame, validation\_set\_percent, test\_set\_percent)

# объекты для подгрузки данных во время обучения с жесткого диска

dataloaders = []

for dataset in datasets:

dataloaders.append(create\_dataloader(data\_frame, image\_data\_generator, image\_size))

# скомпилированная модель

model = compile\_model(classname)

# история обучения модели

model\_training\_history = train(model, dataloaders[0], dataloaders[1])

save\_model(model)

save\_plots(model\_training\_history, classname)

# результаты тестирования модели

test\_results = test\_model(model, dataloaders[2])

save\_test\_results(test\_results, classname)

# аргументы командной строки

args = get\_cmd\_args()

# CSV-файл

csv\_file = load\_csv(args[0])

if is\_csv\_valid(csv\_file):

if len(args) > 1:

synthesize\_model(csv\_file, args[1])

else:

for disease in csv\_file.diseases():

synthesize\_model(csv\_file, disease)

print("Синтезирование было успешно завершено")

else:

print("CSV-файл не соответствует заявленным требованиям")

**Класс ModelTemplate**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine.utils;

import ai.djl.MalformedModelException;

import ai.djl.inference.Predictor;

import ai.djl.modality.cv.Image;

import ai.djl.modality.cv.transform.Resize;

import ai.djl.ndarray.NDArray;

import ai.djl.ndarray.NDManager;

import ai.djl.repository.zoo.Criteria;

import ai.djl.repository.zoo.ModelNotFoundException;

import ai.djl.repository.zoo.ModelZoo;

import ai.djl.repository.zoo.ZooModel;

import ai.djl.translate.TranslateException;

import java.io.IOException;

/\*\*

\* Инкапсулирует в себе работу с моделью

\*/

public class ModelTemplate {

// Менеджер для работы с массивами

private static final NDManager ndManager;

// Массив со значениями стандартного отклонения для ImageNet

private static final NDArray standardDeviation;

// Массив со значениями среднего для ImageNet

private static final NDArray mean;

// Преобразователь изображения в вещественное число

private static final ImageFloatTranslator imageFloatTranslator;

static {

ndManager = NDManager.newBaseManager();

standardDeviation = ndManager.create(new float[]{0.485f, 0.456f, 0.406f});

mean = ndManager.create(new float[]{0.229f, 0.224f, 0.225f});

imageFloatTranslator =

ImageFloatTranslator.builder()

.addTransform(new Resize(256, 256))

.addTransform(array -> array

.div(255f)

.sub(standardDeviation)

.div(mean)

.div(255f))

.build();

}

// Объект, выдающий предсказание

private Predictor<Image, Float> predictor;

/\*\*

\* Конструктор

\* @param modelPath путь к модели

\*/

public ModelTemplate(String modelPath) throws MalformedModelException, ModelNotFoundException, IOException {

// Шаблон для загрузки модели

Criteria<Image, Float> criteria = Criteria.builder()

.setTypes(Image.class, Float.class)

.optTranslator(imageFloatTranslator)

.optModelUrls(String.format("file:///%s", modelPath))

.build();

// Загруженная модель

ZooModel<Image, Float> model = ModelZoo.loadModel(criteria);

predictor = model.newPredictor();

}

/\*\*

\* Выдача предсказания по изображению

\* @param image изображение

\* @return предсказание

\*/

public Float predict(Image image) throws TranslateException {

return predictor.predict(image);

}

}

**Класс ImageFloatTranslator**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine.utils;

import ai.djl.modality.cv.translator.BaseImageTranslator;

import ai.djl.ndarray.NDList;

import ai.djl.translate.TranslatorContext;

/\*\*

\* Вспомогательный класс для фреймворка Deep Java Library.

\* Позволяет передавать в модель изображение, а на выходе получать результат в виде вещественного числа

\*/

public class ImageFloatTranslator extends BaseImageTranslator<Float> {

/\*\*

\* Конструктор

\*

\* @param builder строитель

\*/

public ImageFloatTranslator(BaseBuilder<?> builder) {

super(builder);

}

/\*\*

\* Создает новый объект, реализующий шаблон проектирования "Строитель"

\* @return объект, реализующий шаблон проектирования "Строитель"

\*/

public static Builder builder() {

return new Builder();

}

/\*\*

\* Обрабатывает выходное значение

\* @param ctx контекст

\* @param list список значений

\* @return предсказание

\* @throws Exception

\*/

@Override

public Float processOutput(TranslatorContext ctx, NDList list) throws Exception {

return list.get(0).getFloat(0);

}

/\*\*

\* Класс-строитель

\*/

public static class Builder extends BaseBuilder<Builder> {

/\*\*

\* Возвращает самого себя

\* @return строитель

\*/

@Override

protected Builder self() {

return this;

}

/\*\*

\* Создает строитель

\* @return строитель

\*/

public ImageFloatTranslator build() {

validate();

return new ImageFloatTranslator(this);

}

}

}

**Класс DiseasePredictorResult**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine;

/\*\*

\* Инкапсулирует в себе сведения о вероятности присутствия определенного заболевания

\*/

public class DiseasePredictionResult {

// Название болезни

private String diseaseName;

// Предсказание

private Float result;

/\*\*

\* Конструктор

\* @param diseaseName название болезни

\* @param result результат

\*/

public DiseasePredictionResult(String diseaseName, Float result) {

this.diseaseName = diseaseName;

this.result = result;

}

// Возвращает название болезни

public String getDiseaseName() {

return diseaseName;

}

// Возвращает предсказание

public Float getResult() {

return result;

}

}

**Класс DiseasePredictor**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine;

import ai.djl.modality.cv.Image;

import ai.djl.translate.TranslateException;

import com.golovkin.chestxraydetector.engine.utils.ModelTemplate;

/\*\*

\* Используется, чтобы диагностировать заболевание, используя переданный ему в качестве аргумента конструктора ModelTemplate

\*/

public class DiseasePredictor {

// название болезни

private String name;

// шаблон модели

private ModelTemplate modelTemplate;

/\*\*

\* Конструктор

\* @param name название болезни

\* @param modelPath шаблон модели

\*/

public DiseasePredictor(String name, String modelPath) {

this.name = name;

try {

modelTemplate = new ModelTemplate(modelPath);

}

catch (Exception e) {

e.printStackTrace();

}

}

/\*\*

\* Предсказывает вероятность присутствия болезни на рентгенограмме

\* @param image изображение рентгенограммы

\* @return вероятность присутствия болезни на рентгенограмме

\*/

public Float predict(Image image) throws TranslateException {

return modelTemplate.predict(image);

}

// Возвращает название болезни

public String getName() {

return name;

}

}

**Класс XrayValidator**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine;

import ai.djl.MalformedModelException;

import ai.djl.modality.cv.Image;

import ai.djl.repository.zoo.ModelNotFoundException;

import ai.djl.translate.TranslateException;

import com.golovkin.chestxraydetector.engine.utils.ModelTemplate;

import java.io.IOException;

/\*\*

\* Класс для проверки на то, что на изображении действительно рентгенограмма

\*/

public class XrayValidator {

// шаблон модели

private ModelTemplate modelTemplate;

/\*\*

\* Конструктор

\* @param modelPath путь к модели

\*/

public XrayValidator(String modelPath) {

try {

modelTemplate = new ModelTemplate(modelPath);

} catch (MalformedModelException | ModelNotFoundException | IOException e) {

e.printStackTrace();

}

}

/\*\*

\* Проверяет, действительно ли на изображении рентгенограмма

\* @param image изображение

\* @return действительно ли на изображении рентгенограмма

\*/

public boolean isValidXray(Image image) throws TranslateException {

Float prediction = modelTemplate.predict(image);

return prediction >= 0.9;

}

}

**Класс CheckXrayController**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.controllers;

import ai.djl.modality.cv.Image;

import com.golovkin.chestxraydetector.engine.DiseasePredictionResult;

import com.golovkin.chestxraydetector.engine.DiseasePredictor;

import com.golovkin.chestxraydetector.engine.XrayValidator;

import com.golovkin.chestxraydetector.engine.utils.ImageIoUtils;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.multipart.MultipartFile;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import org.springframework.web.servlet.view.RedirectView;

import java.util.ArrayList;

import java.util.Base64;

import java.util.Comparator;

import java.util.List;

/\*\*

\* Контроллер веб-приложения, который содержит в себе логику для корректной работы с HTTP-запросами пользователей, направленных на получение рекомендаций от модели (загрузка рентгенограммы, её обработка и т.д.)

\*/

@Controller

public class CheckXrayController {

// Предсказания с результатом 0%

private List<DiseasePredictionResult> emptyPredictions;

// Предсказатели заболеваний

private List<DiseasePredictor> diseasePredictors;

// Объект, проверяющий, действительно ли на изображении рентгенограмма

private XrayValidator xrayValidator;

// Объект для сортировки результатов диагностики по убыванию вероятности

private Comparator<DiseasePredictionResult> comparator;

/\*\*

\* Конструктор

\*

\* @param diseasePredictors предсказатели заболеваний

\* @param xrayValidator объект для сортировки результатов диагностики по убыванию вероятности

\* @param comparator объект для сортировки результатов диагностики по убыванию вероятности

\*/

public CheckXrayController(List<DiseasePredictor> diseasePredictors, XrayValidator xrayValidator, Comparator<DiseasePredictionResult> comparator) {

this.diseasePredictors = diseasePredictors;

this.xrayValidator = xrayValidator;

this.comparator = comparator;

emptyPredictions = new ArrayList<>();

for (DiseasePredictor diseasePredictor : diseasePredictors) {

emptyPredictions.add(new DiseasePredictionResult(diseasePredictor.getName(), 0f));

}

}

/\*\*

\* Обработка GET-запроса по пути <URL веб-приложения>/checkXray

\*

\* @param model модель (MVC)

\* @return имя представления

\*/

@GetMapping("checkXray")

@SuppressWarnings("unchecked")

public String checkXray(Model model) {

List<DiseasePredictionResult> predictions = (List<DiseasePredictionResult>) model.getAttribute("predictions");

boolean isEmptyPredictions = false;

if (predictions == null) {

predictions = emptyPredictions;

isEmptyPredictions = true;

} else {

predictions.sort(comparator);

}

model.addAttribute("predictions", predictions);

model.addAttribute("isEmptyPredictions", isEmptyPredictions);

return "index";

}

/\*\*

\* Обработка POST-запроса по пути <URL веб-приложения>/predictDiseases

\*

\* @param multipartFile загруженный файл

\* @param redirectAttributes атрибуты для реализации паттерна Post-Redirect-Get

\* @return перенаправление на представление

\*/

@PostMapping("predictDiseases")

public RedirectView predictDiseases(@RequestParam("file") MultipartFile multipartFile, RedirectAttributes redirectAttributes) {

try {

byte[] imageAsByteArray = ImageIoUtils.getFileAsByteArray(multipartFile);

Image image = ImageIoUtils.getImage(imageAsByteArray);

boolean isValidXray = xrayValidator.isValidXray(image);

if (isValidXray) {

List<DiseasePredictionResult> predictions = new ArrayList<>();

for (DiseasePredictor diseasePredictor : diseasePredictors) {

String diseaseName = diseasePredictor.getName();

Float prediction = diseasePredictor.predict(image);

predictions.add(new DiseasePredictionResult(diseaseName, prediction));

}

redirectAttributes.addFlashAttribute("imageAsByte64", "data:image/\*;base64, " + Base64.getEncoder().encodeToString(imageAsByteArray));

redirectAttributes.addFlashAttribute("predictions", predictions);

} else {

throw new IllegalArgumentException("Загруженная фотография - не рентгенограмма");

}

} catch (Exception e) {

RuntimeException newException = new RuntimeException("Не удалось обработать рентгенограмму", e);

throw newException;

}

return new RedirectView("checkXray");

}

}

**Класс ImageUtils**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.engine.utils;

import java.awt.\*;

import java.awt.image.BufferedImage;

/\*\*

\* Класс, предназначенный для проверки изображения на соответствие заданным требованиям (размер не менее 256x256 пикселей) и приведения их к ним (создание изображения с соотношением сторон 1:1 из исходного)

\*/

public class ImageUtils {

// Минимальный размер загружаемого изображения

private static final int MIN\_IMAGE\_SIZE = 256;

/\*\*

\* Проверяет, действительно ли изображение больше или равно минимальному размеру загружаемого изображения

\* @param bufferedImage изображение

\* @return действительно ли изображение больше или равно минимальному размеру загружаемого изображения

\*/

public static boolean checkSize(BufferedImage bufferedImage) {

if (bufferedImage.getWidth() < MIN\_IMAGE\_SIZE && bufferedImage.getHeight() < MIN\_IMAGE\_SIZE) {

throw new IllegalArgumentException(String.format("Загруженная фотография должна иметь размер не меньше %dx%d"), MIN\_IMAGE\_SIZE, MIN\_IMAGE\_SIZE);

} else {

return true;

}

}

/\*\*

\* Проверяет, действительно ли изображение имеет соотношение 1:1

\* @param bufferedImage изображение

\* @return действительно ли изображение имеет соотношение 1:1

\*/

public static boolean checkRatio(BufferedImage bufferedImage) {

return bufferedImage.getWidth() != bufferedImage.getHeight();

}

/\*\*

\* Получение изображения с соотношением сторон 1:1

\* @param image изображение

\* @return изображение с соотношением сторон 1:1

\*/

public static BufferedImage getSquaredImage(BufferedImage image) {

int maxSize = Math.max(image.getWidth(), image.getHeight());

BufferedImage newImage = new BufferedImage(maxSize,maxSize, BufferedImage.TYPE\_INT\_RGB);

Graphics graphics = newImage.getGraphics();

graphics.setColor(Color.black);

graphics.fillRect(0, 0, maxSize,maxSize);

graphics.drawImage(image, 0, 0, null);

graphics.dispose();

return newImage;

}

}

**Класс SendNewDataController**

// Тема ВКР: Разработка программного обеспечения для диагностики заболеваний дыхательной системы человека по рентгенограмме грудной клетки

// Автор: Головкин Н.В., 09.03.04, 743

// Научный руководитель: Цуканова Н.И., доцент

// Средства разработки: Java 8, Spring Framework 5.3.7, Deep Java Library 0.11.0

// Назначение данной части ПО: Класс из веб-приложения

// Дата создания: 15.05.2021

package com.golovkin.chestxraydetector.controllers;

import ai.djl.modality.cv.Image;

import com.golovkin.chestxraydetector.engine.DiseasePredictionResult;

import com.golovkin.chestxraydetector.engine.DiseasePredictor;

import com.golovkin.chestxraydetector.engine.XrayValidator;

import com.golovkin.chestxraydetector.engine.utils.ImageIoUtils;

import org.springframework.stereotype.Controller;

import org.springframework.ui.Model;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestParam;

import org.springframework.web.multipart.MultipartFile;

import org.springframework.web.servlet.mvc.support.RedirectAttributes;

import org.springframework.web.servlet.view.RedirectView;

import java.util.ArrayList;

import java.util.List;

/\*\*

\* Контроллер веб-приложения, который содержит в себе логику для отправки на компьютер, предназначенный для обучения модели, новых рентгенологических исследований

\*/

@Controller

public class SendNewDataController {

// Объект, проверяющий, действительно ли на изображении рентгенограмма

private XrayValidator xrayValidator;

// Конфигурация NFS сервера

private NfsServerConfig nfsServerConfig;

// Предсказатели заболеваний

private List<DiseasePredictor> diseasePredictors;

/\*\*

\* Конструктор

\*

\* @param diseasePredictors предсказатели заболеваний

\* @param xrayValidator объект, проверяющий, действительно ли на изображении рентгенограмма

\* @param nfsServerConfig конфигурация NFS сервера

\*/

public SendNewDataController(List<DiseasePredictor> diseasePredictors, XrayValidator xrayValidator, NfsServerConfig nfsServerConfig) {

this.diseasePredictors = diseasePredictors;

this.xrayValidator = xrayValidator;

this.nfsServerConfig = nfsServerConfig;

}

/\*\*

\* Обработка GET-запроса по пути <URL веб-приложения>/sendNewData

\*

\* @param model модель (MVC)

\* @return имя представления

\*/

@GetMapping("sendNewData")

@SuppressWarnings("unchecked")

public String showSendNewDataPage(Model model) {

model.addAttribute("diseasePredictors", diseasePredictors);

return "sendNewData";

}

/\*\*

\* Обработка POST-запроса по пути <URL веб-приложения>/sendNewData

\*

\* @param multipartFile загруженный файл

\* @param diseases отмеченные заболевания

\* @param redirectAttributes атрибуты для реализации паттерна Post-Redirect-Get

\* @return перенаправление на представление

\*/

@PostMapping("sendNewData")

public RedirectView sendNewData(@RequestParam("file") MultipartFile multipartFile, List<DiseasePredictionResult> diseases, RedirectAttributes redirectAttributes) {

try {

byte[] imageAsByteArray = ImageIoUtils.getFileAsByteArray(multipartFile);

Image image = ImageIoUtils.getImage(imageAsByteArray);

boolean isValidXray = xrayValidator.isValidXray(image);

if (isValidXray) {

List<DiseasePredictionResult> predictions = new ArrayList<>();

for (DiseasePredictor diseasePredictor : diseasePredictors) {

String diseaseName = diseasePredictor.getName();

Float prediction = diseasePredictor.predict(image);

predictions.add(new DiseasePredictionResult(diseaseName, prediction));

}

try {

NfsServerUtils.sendImage(image, diseases);

redirectAttributes.addFlashAttribute("message", "Результаты были успешно отправлены");

redirectAttributes.addFlashAttribute("messageStatus", MessageStatus.SUCCESS);

} catch (Exception e) {

redirectAttributes.addFlashAttribute("message", "Не удалось отправить результаты рентгенологического исследования. Пожалуйста, попробуйте позже");

redirectAttributes.addFlashAttribute("messageStatus", MessageStatus.ERROR);

}

} else {

throw new IllegalArgumentException("Загруженная фотография - не рентгенограмма");

}

} catch (Exception e) {

RuntimeException newException = new RuntimeException("Не удалось обработать рентгенограмму", e);

throw newException;

}

return new RedirectView("sendNewData");

}

}