Федеральное государственное автономное образовательное учреждение высшего образования

«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Гуманитарный институт

Кафедра фундаментальной и прикладной лингвистики

КУРСОВАЯ РАБОТА

на тему

**«ПРИМЕНЕНИЕ И СРАВНЕНИЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ ДЛЯ АВТОМАТИЧЕСКОЙ ОЦЕНКИ ОТЗЫВОВ НА ФИЛЬМЫ И СЕРИАЛЫ»**

Выполнил:

Студент 2 курса специализации

«Фундаментальная и прикладная лингвистика»

очной формы обучения

Шаймухамедов Александр Александрович

Руководитель:

Ассистент кафедры ФиПЛ

Сартаков Артемий Михайлович

Новосибирск, 2025 год

**Содержание**

Введение 2

Глава I. Теоретическая часть 5

1.1 Классические методы классификации текстов 5

1.2 Методы классификации текстов с использованием нейронных сетей 9

Глава II. Практическая часть 12

2.1 Выбор инструментов 12

2.2 Структура проекта 14

2.3 Создание датасета 15

2.4 Применение классических методов 19

2.5 Применение нейронных сетей 21

Заключение 27

Библиографический список 29

Приложение А. Код программы 30

Приложение B.1. Структура проекта 30

Приложение B.2. Результаты работы нейросетей 31

Приложение B.3. Работа программы 32

Введение

Классификация текста – это процесс определения и присвоения одной из предопределенных категорий тексту с опорой на его содержимое. Задача классификации текста (Text Classification) является одной из самых интересных, сложных и актуальных задач такого направления в области компьютерной лингвистики и машинного обучения, как обработка естественного языка (Natural Language Processing, NLP). Для решения этой задачи существует множество подходящих методов, каждый из которых отличается от других не только по требовательности к ресурсам, скорости и производительности, но и по конкретному типу задачи, к которому наиболее хорошо применим. [1]

Целью данной курсовой работы является создание программы, которая способна определять, какой рейтинг вероятнее всего пользователь присвоил бы текстовому отзыву на фильм или сериал на английском языке.

Для достижения данной цели были поставлены следующие задачи:

1. Изучить теоретические материалы, связанные с различными методами машинного обучения, используемыми в задаче классификации текста;
2. Создать датасет на основе нескольких тысяч оцененных отзывов на фильмы и сериалы с веб-сайта https://imdb.com;
3. Написать код для обучения, оценки и сравнения классических и нейронных моделей в задаче мультиклассовой классификации текстов;
4. Выбрать наиболее подходящую для поставленной задачи модель и сохранить ее;
5. Написать программу для использования лучшей модели;
6. Закончить программу, обеспечив ее корректную работу;
7. Провести тестирование программы;
8. Оформить конечный код по стандарту PEP 8.

Настоящая курсовая работа состоит из введения; двух глав в основной части, первая из которых состоит из двух параграфов, вторая же – из пяти; заключения и библиографического списка, а также приложения со ссылкой на репозиторий проекта на GitHub и иллюстративные материалы. Во введении сформулирована актуальность работы, поставлена цель и описаны задачи, необходимые для ее достижения, а также представлена структура работы. В Главе I «Теоретическая часть» дается общий обзор классических методов и методов, использующих нейронные сети, в задаче классификации текстов. В Главе II «Практическая часть» описаны выбранные инструменты, структура проекта, а также даются пояснения к работе всех файлов проекта. В заключении подводятся итоги проведенной работы.

Глава I. Теоретическая часть

1.1 Классические методы классификации текстов

Классификация текстов в общем случае происходит на основе корпуса текстов, каждый из которых уже имеет свой класс, который далее будем также называть лейблом, следующим образом: в качестве ввода программе-классификатору подается текст, а в качестве вывода программа присваивает этому тексту наиболее вероятный из возможных лейбл. В текущей работе мы рассматриваем лишь те методы, которые позволяют получить единственный «правильный» лейбл.

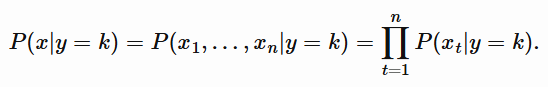
Работа программы-классификатора состоит из двух блоков: выделения признаков (feature extraction) и собственно классификации. Выделение признаков может быть определено вручную (эти методы будут рассмотрены далее) или выучено (с помощью нейронных сетей, о таких методах в следующей части). Классификатор должен присвоить вероятности для каждого класса, основываясь на полученном в результате выделения признаков представлении текста, и выбрать самый вероятный класс.

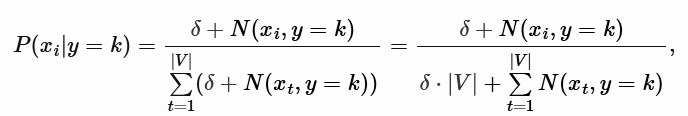
Классические методы классификации текстов были развиты задолго до того, как нейронные сети стали популярными, и до сих пор имеют высокую эффективность при сравнительно небольших размерах датасетов.

В первую очередь рассмотрим наивный байесовский классификатор. Он основан на применении теоремы Байеса со строгими (наивными) предположениями об условной независимости: предполагается, что слова в текстах никак не влияют друг на друга, то есть не важен ни контекст, ни порядок слов. Достоинствами этого метода можно назвать небольшое количество данных, которые нужны для обучения, оценки параметров и классификации.

Наивный байесовский классификатор совершает предсказание о лейбле следующим образом:

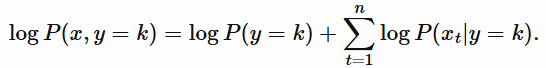
«P(y = k)», то есть вероятность того, что лейбл равен «k», считается с помощью метода максимального правдоподобия делением количества текстов класса «k» на суммарное количество текстов всех возможных классов в датасете.

Условная же вероятность «P(x | y = k)» считается следующим образом:

В формуле выше условная вероятность для каждого токена «xi» текста «x», где токен – наименьшая единица текста, которую обрабатывает модель (слово или пунктуационный знак), считается следующим образом:

где дельта определяется вручную (зачастую ее значения близки к единице) и необходима для того, чтобы избежать деление на ноль в случае отсутствия слова во всех текстах, на которых обучена модель.

На практике чаще всего используется логарифмическая вероятность, поскольку интерес представляет лишь argmax:



Следующей моделью, которую мы рассмотрим, является классификатор максимальной энтропии (логистическая регрессия). Этот метод требует также вручную выделенные признаки, но классифицирует иначе.

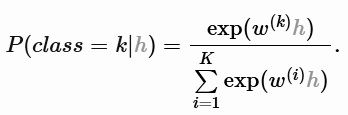
В текущей работе на практике мы будем применять такие методы выделения признаков для последующей логистической регрессии, как «Tf-Idf» и «Word Vectors».

«Tf-Idf» (Text frequency / Inverse document frequency) вычисляется следующим образом для каждого признака:

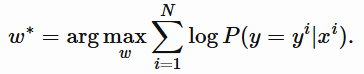
featurei = Count(wordi ∈ x) × log(N / (Count(wordi ∈ D) + α)), где x – один текст, D – датасет (коллекция текстов, или документов), N – общее число документов в датасете, α – гиперпараметр сглаживания (обычно равен 1), Count(wordi ∈ x) – число текстов, в которых встречается wordi.

«Word Vectors» – еще один подход к выделению признаков, который вместо частотности каждого отдельного токена рассматривает предобученные векторы, что позволяет сокращать количество признаков.

Общий конвейер классификации логистической регрессии выглядит следующим образом:

1. Получить «h = (f1, f2, …, fn)» – представление признаков входного текста;
2. Взять «w(i) = (w(i)1, …, w(i)n)» – векторы с весами признаков для каждого из классов;
3. Для каждого класса взвесить признаки, то есть взять скалярное произведение представлений признаков «h» с весами характеристик «w(k)», при этом «f0» считается равной 1:
4. Получить вероятности классов с помощью softmax (многопеременной логистической функции):

Softmax при этом нормализует K-значения, которые получаются на предыдущем этапе распределения вероятностей, по выходным классам.

Параметры «w\*» выбираются таким образом, чтобы максимизировать вероятность обучающих данных:

Чтобы найти параметры, максимизирующие логарифмическое правдоподобие данных, используется градиентный подъем: веса постепенно улучшаются в ходе нескольких итераций по данным, поскольку при каждой итерации максимизируется вероятность, которую модель назначает правильному классу.

В заключение этой части стоит теоретически сопоставить рассмотренные модели классификаторов. Оба метода довольно просты и интерпретируемы, а также используют ручное выделение признаков. Тем не менее, логистическая регрессия обучается медленнее за счет многократного прохождения через данные, а наивный байес предполагает, что признаки условно независимы для данного класса, чего не делает логистическая регрессия. Таким образом, оба рассмотренных метода по-своему хороши в решении прикладных задач. [2]

1.2 Методы классификации текстов с использованием нейронных сетей

Классические методы доказали свою эффективность во многих задачах, особенно при работе с небольшими наборами данных или в условиях ограниченных вычислительных ресурсов. Тем не менее, они ограничены в способности улавливать сложные семантические и синтаксические зависимости в тексте. Нейронные сети предлагают принципиально другой подход к анализу: они автоматически извлекают признаки из данных, учатся на их распределении и способны моделировать сложные нелинейные зависимости. Благодаря архитектурам, специально разработанным для обработки текстов, нейросетевые методы демонстрируют высокую точность в задаче классификации текстов, в особенности на значительных объемах данных. Однако, за это приходится платить повышенными требованиями к вычислительным ресурсам и сложностью в интерпретации результатов. В этой главе мы рассмотрим основы ключевых архитектур нейронных сетей, которые используются в задаче классификации текстов. [1]

Для начала рассмотрим рекуррентные нейронные сети (RNNs, Recurrent Neural Networks). Они, подобно людям, поочередно «читают» последовательность токенов и обрабатывают информацию: связи между элементами образуют направленную последовательность. На каждом шагу рекуррентная модель принимает новый входной вектор (например, эмбеддинг токена) и предыдущее скрытое состояние модели. Используя входной вектор, RNN-клетка, одинаковая для каждого шага, получает новое скрытое состояние, которое содержит информацию о текущем векторе и предыдущих шагах. [2]



Самые простые рекуррентные модели имеют лишь один слой. При выборе такой архитектуры следует использовать последнее скрытое состояние RNN-клетки, поскольку именно в нем она «видела» весь текст. Зачастую для лучшего представления текста используют несколько слоев. При таком подходе входными данными для клетки более высоких слоев на каждом шаге выступают не непосредственные эмбеддинги токенов, а состояния клетки предыдущего слоя на том же шаге. Предполагается, что в такой модели более низкие слои будут «понимать» местные феномены (например, фразы), а более высокие – более сложные вещи, например, тему текста.

Такие простые сети называются ванильными RNN и подходят для изучения краткосрочных зависимостей, но ограничены проблемой исчезающих градиентов. [3] Для преодоления этой проблемы, Long Short-Term Memory Networks (LSTMs, сети с долговременной и кратковременной памятью, подтип RNN) используют механизм памяти. Каждая клетка LSTM имеет три вентиля (gates). Входной вентиль контролирует, сколько новой информации следует добавить к текущему состоянию клетки. Вентиль забывания определяет, от какой предыдущей информации можно избавиться. Выходной вентиль регулирует, какая информация должна возвращаться на текущем шаге. Такая выборочная память позволяет LSTM поддерживать долгосрочные зависимости, делая их идеальным решением для задач, в которых критичен ранний контекст.

Gated Recurrent Unit (GRU, управляемый рекуррентный блок) упрощает работу LSTM, объединяя вентили входа и забывания в единый вентиль обновления и рационализируя механизм выхода. Эта архитектура вычислимо-эффективна и близка по производительности к LSTM, что полезно в задачах, где простота и скорость обучения являются преимуществом. [4]

В конце концов, рассмотрим сверточные нейронные сети (CNNs, Convolutional Neural Networks). Изначально они были разработаны для задач компьютерного зрения, то есть распознавания образов путем разбиения изображения на множество частей и поиска нужного паттерна на каждой из них. Подобно такому подходу, эти модели могут быть применены и в задаче классификации текстов: нам нужно находить определенные паттерны (например, словосочетания), при этом совершенно неважно, в какой части текста они находятся.

Для осуществления такого подхода CNN имеет два особенных слоя: слой свертки, который находит совместимости с паттернами, пробегая по токенам «окном» определенного размера, и слой субдискретизации (или «пулинга»), который фильтрует нерелевантные признаки, оставляя лишь найденные соответствия. В задаче классификации текста обычно используется «макспулинг», который сжимает все полученные после применения слоя свертки и нелинейности (ReLU) векторы в один вектор-репрезентатив текста, который далее, как и в остальных нейронных сетях, проходит через линейный слой, и с помощью softmax определяется предсказание для класса. [5]

Глава II. Практическая часть

2.1 Выбор инструментов

Для разработки программы были выбраны следующие параметры:

* Язык программирования – Python. Выбор языка был обусловлен поддержкой всех необходимых библиотек, наличием удобных встроенных модулей, а также простотой и скоростью разработки;
* Версия языка программирования – Python 3.13;
* Среда разработки – PyCharm Community Edition 2024.1.3 – одна из наиболее удобных сред для разработки на Python;
* Архитектура параллельных вычислений для работы нейронных сетей – CUDA Toolkit 11.8. Выбор версии был обусловлен видеокартой NVIDIA GeForce GTX 1060 и версией драйверов 472.12.

Далее рассмотрим все встроенные модули Python (Таблица 1) и сторонние библиотеки (Таблица 2), которые были использованы в процессе разработки программы.

Таблица 1 – используемые встроенные модули Python и их описания:

|  |  |
| --- | --- |
| **Модуль** | **Описание** |
| collections[[1]](#footnote-1) | Предоставляет специализированные контейнеры типов данных, альтернативные встроенным в Python контейнерам типов данных общего назначения. В работе используется Counter – подкласс словаря для подсчета хешируемых объектов. |
| csv[[2]](#footnote-2) | Предоставляет возможности для работы с Comma Separated Values – самым распространенным форматом таблиц и баз данных. |
| json[[3]](#footnote-3) | Предоставляет возможности для кодирования Python-объектов в JSON-объекты и/или их декодирования. |
| time[[4]](#footnote-4) | Предлагает функции, связанные со временем. В работе используется time.sleep() для пауз между запросами к веб-страницам при парсинге. |

Таблица 2 – используемые сторонние библиотеки и их описания:

|  |  |
| --- | --- |
| **Библиотека** | **Описание** |
| gensim[[5]](#footnote-5) | Библиотека для использования методов обработки искусственного языка при обработке больших текстовых данных. В работе используется для загрузки предобработанных эмбеддингов (векторов) в классе WordVectorsModel. |
| matplotlib[[6]](#footnote-6) | Библиотека, позволяющая визуализировать данные. |
| nltk[[7]](#footnote-7) | Natural Language Toolkit – пакет библиотек и программ для символьной и статистической обработки естественного языка. В работе используется метод WordPunctTokenizer из nltk.tokenize для качественной токенизации текстов. |
| numpy[[8]](#footnote-8) | Библиотека, позволяющая работать с многомерными векторными массивами и применять высокоуровневые математические операции. В работе используется для выполнения операций над векторами и матрицами. |
| pandas[[9]](#footnote-9) | Библиотека для обработки и анализа табличных данных. В работе используется для чтения csv-файлов. |
| selenium[[10]](#footnote-10) | Инструмент для автоматизации действий веб-браузера. В работе используется для парсинга динамических страниц в файле create\_dataset.py. |
| sklearn[[11]](#footnote-11) | Scikit-learn – библиотека, предназначенная для машинного обучения. В работе используются: accuracy\_score и ckassification\_report для оценки моделей, LogisticRegression и MultinomialNB для создания моделей логистической регрессии и наивного байесовского классификатора по нескольким классам. |
| torch[[12]](#footnote-12) | Основной фреймворк для машинного обучения, использующая собственную структуру данных – n-мерные тензоры. В работе используется при работе с нейронными сетями. |

2.2 Структура проекта

Этапы работы программы можно условно разделить на 2 части: подготовительную и актуальную.

Первая часть названа подготовительной, поскольку ее выполнение необходимо для реализации актуальной части работы. Во время выполнения подготовительной части происходит сбор данных для формирования и непосредственное формирование датасета из отзывов на фильмы и сериалы с сайта IMDb[[13]](#footnote-13). К этой части относятся такие файлы, как:

* «create\_dataset.py», в котором происходит работа с веб-страницами, создается и актуализируется база данных в виде csv-файлов «train\_reviews.csv» и «test\_reviews.csv»;
* «metrics.py», в котором происходит оценка параметров созданного датасета.

К актуальной части относятся следующие файлы:

* «classical\_methods.py» – файл с кодом для реализации и сравнения производительности рассмотренных в теоретической части классических методов для классификации текстов: наивного байесовского классификатора с использованием Bag Of Words и логистической регрессии с использованием Tf-Idf и Word Vectors;
* «neural\_networks.py» – файл с кодом для реализации, сравнения производительности и ее визуализации рассмотренных в теоретической части нейронных сетей для классификации текстов: RNN, LSTM, GRU и TextCNN;
* «application.py» – файл для использования моделей.

Всю структуру проекта можно увидеть в Приложении B.1.

2.3 Создание датасета

Было принято решение создавать датасет из оцененных отзывов с сайта IMDb. Это решение было обусловлено тем, что IMDb – Internet Movie Database, интернет-база данных о фильмах – не только не ограничивает парсинг своих страниц в некоммерческих целях, но и имеет миллионы отзывов на фильмы и сериалы. Более того, этот сайт предоставляет общедоступный датасет из пятидесяти тысяч предобработанных отзывов: по двадцать пять тысяч для обучения и оценки моделей. Тем не менее, этот датасет предлагает бинарную классификацию: в нем представлены условно-плохие и условно-хорошие отзывы, а оригинальные оценки пользователей не сохранены. В связи с этим, такое готовое решение, несмотря на простоту его использования, оказалось недостаточным для нашей цели осуществить мультиклассовую классификацию.

Создание файлов «train\_reviews.csv» и «test\_reviews.csv» происходит в файле «create\_dataset.py».

В строках 3–11 этого файла указаны импортируемые модули и библиотеки. В строках 14–22 описана функция setup\_driver() для подключения к драйверу для парсинга веб-страниц. В строках 25–34 описывается функция get\_reviews\_pages(driver, url), которая принимает драйвер и ссылку на тематическую коллекцию с сайта IMDb и возвращает список со ссылками на страницы с отзывами на каждое из произведений из коллекции. Решение использовать уже готовые тематические коллекции было принято в целях более простого отбора произведений. Были использованы такие коллекции, как «Худшие сериалы»[[14]](#footnote-14), «Лучшие сериалы»[[15]](#footnote-15) и «Худшие фильмы»[[16]](#footnote-16).

В функции expand\_spoilers(driver) (строки 37–52) описан алгоритм раскрытия спойлеров на текущей итерации загрузки веб-страницы. Написание этой функции было обусловлено тем, что многие отзывы обычно маркируются пометкой «Spoiler», и их текст не только не отображается на сайте при ручном скроллинге веб-страницы, но и не содержится в коде этой страницы, а потому для получения текста отзыва необходимо нажимать на кнопку с пометкой. По этой причине, в частности, для парсинга страниц была использована библиотека Selenium, которая позволяет парсить интерактивные веб-страницы.

Функция scroll\_to\_element(driver, element) позволяет прокручивать страницу к нужному элементу.

Идентичные функции write\_into\_train(review\_rating, review\_text) и write\_into\_test(review\_rating, review\_text) (строки 60–64 и 67–71 соответственно) позволяют записывать данные отзыва в нужный файл.

Функция check\_if\_full(), работающая с глобальными переменными reviews\_amount\_train и reviews\_amount\_test, позволяет проверить, не заполнен ли еще датасет, и используется в основном коде файла.

В строках 83–170 описана основная функция файла – функция для парсинга веб-страниц с отзывами parse\_reviews(driver, url). Она принимает страницу с отзывами на конкретное произведение, циклично читает доступный код страницы, находит в нем все элементы-отзывы, при наличии спойлеров раскрывает их и для каждого отзыва получает его рейтинг и текст, которые попеременно (для равномерного заполнения) записывает в csv-файлы, параллельно инкрементируя значения рейтинга-ключа в одном из словарей (global reviews\_amount\_train, reviews\_amount\_test) для отслеживания заполненности файлов. В конце текущей итерации чтения страницы пытается найти кнопку «More», нажатие на которую позволяет загрузить следующие 25 отзывов, и прокручивает страницу к ним. Работа функции завершается, когда нет возможности найти кнопку «More», что означает, что драйвер дошел до конца текущей страницы. Функция проверяет заполненность датасета с помощью функции check\_if\_full() и возвращает False, если датасет заполнен, и True, если нужны еще отзывы.

В конце файла описана точка входа if \_\_name\_\_ == ‘\_\_main\_\_’ (строки 177–209), которая запускается единожды и в которой создаются csv-файлы с необходимыми заголовками, получаются страницы с отзывами на фильмы и сериалы из коллекций и инициируются словари reviews\_amount\_train и reviews\_amount\_test для отслеживания процесса создания датасета. Итератор for link in all links применяет функцию parse\_reviews(driver, url) для каждой из полученных ссылок с отзывами. Работа программы автоматически завершается при полном заполнении датасета или вручную.

Перед запуском программы была поставлена цель собрать 20 тысяч отзывов для каждого из файла с ограничением по 2 тысячи отзывов на каждый лейбл. К сожалению, эта цель не была достигнута вследствие больших ресурсозатрат, а именно времени: программа непрерывно работала несколько суток, но ей не удалось собрать достаточное количество отзывов со средними оценками несмотря на сравнительно гораздо более быстрое наполнение файлов отзывами с рейтингами 1, 9 и 10. Мы связываем это с тем, что люди стремятся категоризировать свои мнения в терминах «плохо» и «хорошо», вследствие чего отзывы с рейтингами от 2 до 6 наиболее редки даже у произведений из коллекций худших.

В файле «metrics.py» производится оценка размеров полученного датасета. Всего было собрано 16773 отзыва. Приведем итоговые значения словарей reviews\_amount\_train и reviews\_amount\_test для демонстрации полученных результатов работы программы create\_dataset.py:

* Train ratings: {‘1’: 2000, ‘2’: 507, ‘3’: 296, ‘4’: 266, ‘5’: 351, ‘6’: 355, ‘7’: 506, ‘8’: 726, ‘9’: 1408, ‘10’: 2000};
* Test ratings: {‘1’: 2000, ‘2’: 487, ‘3’: 318, ‘4’: 252, ‘5’: 350, ‘6’: 345, ‘7’: 526, ‘8’: 669, ‘9’: 1411, ‘10’: 2000}.

В связи с такими значениями было принято решение о классификации текстов по четырем классам:

1. Одна звезда (2000 отзывов в train и 2000 отзывов в test);
2. От двух до пяти звезд (1420 отзывов в train и 1407 отзывов в test);
3. От шести до восьми звезд (1587 отзывов в train и 1540 отзывов в test);
4. Девять или десять звезд (3408 отзывов в train и 3411 отзывов в test).

В том же файле была осуществлена оценка текстов в датасете. Были получены следующие результаты:

* Длиннейший текст из тренировочных (токенов): 2316;
* Длиннейший текст из тестовых (токенов): 2085;
* Разброс длин тренировочных текстов (по числу токенов): {'0-100': 3599, '101-200': 2756, '201-300': 1043, '301-500': 637, '500+': 380};
* Разброс длин тестовых текстов (по числу токенов): {'0-100': 3538, '101-200': 2712, '201-300': 1077, '301-500': 670, '500+': 361}.

Таким образом, 75,5% тренировочных и 74,8% тестовых отзывов имеют длину от 0 до 200 токенов, в то время как 87,9% тренировочных и 87,7% тестовых отзывов имеют длину от 0 до 300 токенов.

2.4 Применение классических методов

Применение классических методов осуществляется в файле «classical\_methods.py».

В строках 3–9 импортируются необходимые сторонние библиотеки.

В строках 12–21 описывается функция preprocess\_labels(y), которая принимает список лейблов от 1 до 10 и возвращает список с переопределенными лейблами по уже четырем категориям.

В строках 24–39 представлен класс Vocabulary, который позволяет строить частотный словарь слов в текстах.

В строках 42–64 описывается класс MultinomialNaiveBayesModel, который при своей инициализации принимает частотный словарь и инициализирует объект-классификатор MultinomialNB, с помощью которого в методе prediction(...) предсказывает лейблы тестовых текстов. Метод \_text\_to\_bow(text) позволяет извлекать Bag of Words из каждого текста с опорой на частотный словарь.

Строки 67–109 представляют класс TFIDFModel, который так же, как и предыдущий, принимает при своей инициализации частотный словарь, но инициализирует уже объект-классификатор LogisticRegression, с помощью которого в методе prediction(...) предсказывает лейблы тестовых текстов, основываясь на Tf-Idf для каждого текста. Признаки для Tf-Idf извлекаются для каждого текста с помощью метода \_text\_to\_tfidf(text). Он также использует словарь idf\_dict, который считается в методе \_compute\_idf(texts), запускающийся в prediction().

В строках 112–132 описывается класс WordVectorsModel, который тоже использует LogisticRegression, но уже не пользуется частотным словарем, загружая предобработанные эмбеддинги с помощью gensim.downloader.load(‘fasttext-wiki-news-subwords-300’).

В конце файла описана точка входа if \_\_name\_\_ == ‘\_\_main\_\_’ (строки 135–174), которая загружает данные из csv-файлов, создает частотный словарь на основе тренировочных текстов, инициализирует каждую из описанных ранее моделей и с помощью каждой из них предсказывает лейблы для тестовых текстов. Затем происходит оценка предсказаний моделей, результаты экранируются.

В результате работы файла были получены следующие числа:

* Model: NaiveBayes with Manual BOW. Accuracy: 62.9218%;
* Model: LogisticRegression with Manual TF-IDF. Accuracy: 59.6674%;
* Model: LogisticRegression with WordVectors. Accuracy: 62.8021%.

Можем заметить, что все три модели работают примерно одинаково – средне. Тем не менее, наилучший результат был получен наивным байесовским классификатором, что в сумме с его высокой по сравнению с остальными двумя скоростью делает его безоговорочным победителем.

2.5 Применение нейронных сетей

Основная реализация кода, связанного с нейронными сетями, происходит в файле «neural\_networks.py». Для начала рассмотрим его.

Импорт встроенных модулей и сторонних библиотек происходит в строках с 3 по 14.

В строках 17–28 описывается функция preprocess\_labels(y), идентичная той, которая используется в «classical\_methods.py», но использующая лейблы от 0 до 3, поскольку этого требует PyTorch, ибо nn.CrossEntropyLoss при количестве лейблов, равном 4, ожидает их в виде [0, 1, 2, 3].

В строках 31–70 описывается класс ReviewsDataset, который является наследником класса Dataset модуля torch.utils.data. Во время своей инициализации он принимает тексты и лейблы, опционально – словарь и максимальную длину последовательности токенов. Если словарь не передан, то он строится на основе текстов в методе build\_vocab(), при этом первые два индекса (0 и 1) отводятся токенам <PAD> для заполнения последовательностей и <UNK> для неизвестных токенов. Следующие индексы отводятся токенам по убыванию частотности в текстах.

Метод text\_to\_sequence(text) конвертирует текст в список индексов, обрезая до max\_len. По умолчанию max\_len есть 150 токенов, что оптимально для работы нейросетей. Метод len() определяется как количество текстов, а getitem(index) – как словарь из текста в виде его padded-последовательности (то есть дополненная при длине менее 300 токенов токенами <PAD>) текста по его индексу в списке текстов и лейбла текста по его индексу в списке лейблов.

В строках 73–98 описывается класс RNNModel, который при инициализации принимает размер словаря (количество уникальных токенов), а также по умолчанию задает размерность векторного пространства равной 300, размер скрытого состояния равным 512, количество RNN слоев равным 2, количество классов равным 4, вероятность дропаута между слоями равной 0,5 (для предотвращения переобучения) и использует ReLU (Rectified Linear Unit, имеющая формулу ReLU(x) = max(0, x)) в качестве активации для ускорения обучения. Модель инициализирует слой эмбеддингов, RNN слои, Dropout слой и линейный слой, передавая необходимые параметры.

Прямой проход RNN-модели описан в методе forward(x). Входные данные x в формате [batch, seq\_len] проходят через слой эмбеддингов, реформатируясь в [batch, seq\_len, emb\_dim]. Эмбеддинги нормируются с помощью BatchNorm1d и обрабатываются в слоях RNN, которые возвращают rnn\_out (выходы для всех временных шагов [batch, seq\_len, hidden\_dim]) и \_ (последнее скрытое состояние). Из rnn\_out берется только последний временной шаг (rnn\_out[:, -1, :]), к которому применяется dropout. Результат пропускается через полносвязный слой для получения логитов классификации.

В строках со 101 по 124 описывается реализация класса LSTMModel, отличающегося от RNNModel лишь слоями LSTM и тем, что dropout используется только между несколькими слоями. Аналогично с реализацией GRUModel в строках 127–151. Подробнее о работе LSTM и GRU описано в теоретической части.

В строках 154–180 описывается реализация TextCNN, также принимающая размер словаря при инициализации. Помимо типичного слоя эмбеддингов, имеет convs – список 1D-сверточных слоев разных размеров. filter\_sizes = [2, 3, 4, 5], значит, 4 свертки: «окна» на 2, 3, 4 и 5 токенов. Каждая свертка имеет num\_filters = 200, значит, извлекает 200 признаков на окно. Также реализованы типичный dropout-слой и линейный слой.

Прямой проход CNN-модели описан в методе forward(x). Сперва принимается x в формате [batch\_size, seq\_len] и вследствие прохода через слой эмбеддингов получается embedded формата [batch\_size, seq\_len, emb\_dim], который затем пермутируется в [batch\_size, emb\_dim, seq\_len], что необходимо для совместимости с Conv1d. Затем каждая свертка проходит линейный слой с макспулингом, собираются выходы всех сверток и совершается дропаут. В конце линейный слой преобразует признаки в классы.

В строках 183–241 реализуется функция train\_and\_evaluate(model, train\_loader, test\_loader, model\_name, vocab, max\_len, class\_weights=None, num\_epochs=10), которая выполняет полный цикл обучения и оценки нейросетевой модели для классификации текста. Сперва определяется доступное оборудование (GPU/CPU) и модель переносится на выбранное устройство. Использование GPU критически важно для вычисления вычислений, потому программа выполнялась на персональном компьютере с видеокартой NVIDIA, поддерживающей CUDA. Затем подготавливаются компоненты обучения: функция потерь в виде CrossEntropyLoss с весами классов, являющаяся стандартным выбором для многоклассовой классификации, и оптимизатор Adam – адаптивный алгоритм градиентного спуска, оптимизирующий все параметры модели.

Затем, в цикле обучения, для каждой эпохи и для каждого батча подготавливаются данные, обнуляются градиенты, происходит прямой проход, рассчитываются потери, происходят обратное распространение, обновление весов и агрегация потерь; параллельно все это логируется. Затем модель оценивается: model.eval() отключает dropout, torch.no\_grad() существенно ускоряет вычисления, происходит подготовка данных, прямой проход, преобразование выходов в предсказания и сбор результатов. В конце подсчитываются метрики: accuracy\_score и classification\_report, а также модель сохраняется в папку.

Сохранение модели осуществляется с помощью функции save\_model(model, vocab, max\_len, model\_name, accuracy), описываемой в строках 244–266. В директорию «\saved\_models» сохраняются словарь, состояние модели и вся метаинформация.

Строки 269–333 содержат основной код программы – точку входа if \_\_name\_\_ == ‘\_\_main\_\_’. В этой части кода происходит сбор данных, переопределение лейблов, создаются объекты классов ReviewsDataset и DataLoader, строится словарь, инициализируются модели. Затем все модели проходят обучение и оценку, данные визуализируются, экранируется модель с лучшей производительностью.

В результате выполнения программы с оптимальными параметрами моделей, которые были определены в результате нескольких тестовых запусков, были получены следующие результаты точности предсказаний моделями (визуализацию можно увидеть в Приложении B.2):

* RNN – 27,96%;
* LSTM – 47,88%;
* GRU – 50,91%;
* TextCNN – 65,5%.

Мы связываем неудовлетворительные показатели базовой RNN модели с длинными зависимостями (150 токенов) и проблемой исчезающих градиентов. LSTM и GRU работают лучше, но все еще недостаточно хорошо, вероятно, из-за небольшого объема обучающих данных. GRU может работать лучше LSTM вследствие преобладания текстов средней длины.

TextCNN показал хорошие результаты, близкие к точности классических методов, благодаря способности выявлять локальные признаки («absolutely amazing», «the worst of all»), инвариантности к их позиции и параллельной обработки, позволяющей избежать «забывания».

Рассмотрим файл «application.py», в котором представляется возможным использование лучшей модели, которой оказалась TextCNN. При желании программу можно перенастроить на использование другой из сохранившихся моделей, для этого в строках 3–8 импортируются классы для каждой из модели, и в строках 11–35 функция load\_model(model\_name) загружает необходимую модель.

В строках 38–42 описывается функция предобработки текста.

Выполнение программы описано в строках 45–68: загружается модель, у пользователя запрашивается текст отзыва для предсказания, который предобрабатывается и пропускается через сохраненное состояние модели для оценки лейбла.

Весь код программы можно увидеть в Приложении А, в котором размещена ссылка на GitHub-репозиторий.

Пример работы файла «application.py» можно увидеть в Приложении B.3.

Заключение

В ходе работы над данной курсовой работой, нами была создана программа, которая способна определять, какой рейтинг с большей вероятностью пользователь присвоил бы текстовому отзыву на фильм или сериал на английском языке. Для достижения этого, нами были решены следующие задачи.

Мы изучили теоретические материалы, связанные с различными методами машинного обучения, используемыми в задаче классификации текстов. Мы создали датасет на основе 16773 оцененных отзывов на фильмы и сериалы с веб-сайта IMDb. Нами был написан код для использования классических методов и нейронных моделей, а состояния последних были сохранены для дальнейшего использования. Мы выбрали наиболее точную нейронную модель, которой оказалась CNN, и написали программу для ее использования. Мы обеспечили корректную работу программы, провели ее тестирование и оформили конечный код по стандарту PEP 8.

Благодаря работе над данной курсовой работой, мы познакомились с методами классификации текстов, применяющимися в машинном обучении, а также с методами, необходимыми для сбора данных с динамических веб-страниц. В ходе этого знакомства, мы изучили несколько новых для нас сторонних библиотек и фреймворков Python. Вследствие всего объема работы, нами был получен ценный опыт применения и сравнения точности актуальных методов естественной обработки языка.

В качестве будущих улучшений полученных результатов лучшей нейронной модели, TextCNN, представляются возможными, например, реализация использования предобученных эмбеддингов для повышения качества векторных представлений и добавление фильтров больших размеров для захвата длинных зависимостей в текстах.

Библиографический список

1. Классификация документов [Электронный ресурс]. – URL: <https://habr.com/ru/articles/504744/> (дата обращения: 03.06.2025). – Режим доступа: свободный. – Загл. с экрана.
2. Text Classification [Электронный ресурс]. – URL: [https://lena-voita.github.io/nlp\_course/text\_classification.html](https://lena-voita.github.io/nlp_course/text_classification.html%20) (дата обращения: 03.06.2025). – Режим доступа: свободный. – Загл. с экрана.
3. Vanishing and Exploding Gradients Problems in Deep Learning [Электронный ресурс]. – URL: <https://www.geeksforgeeks.org/vanishing-and-exploding-gradients-problems-in-deep-learning/> (дата обращения: 03.06.2025). – Режим доступа: свободный. – Загл. с экрана.
4. Introduction to Recurrent Neural Networks [Электронный ресурс]. – URL: [https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/](https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/%20) (дата обращения: 03.06.2025). – Режим доступа: свободный. – Загл. с экрана.
5. Introduction to Convolutional Neural Network [Электронный ресурс]. – URL: [https://www.geeksforgeeks.org/introduction-convolution-neural-network/](https://www.geeksforgeeks.org/introduction-convolution-neural-network/%20) (дата обращения: 03.06.2025). – Режим доступа: свободный. – Загл. с экрана.

Приложение А. Код программы

Код программы загружен на GitHub: <https://github.com/vodemah/NSU-Course-Work-2025>

Приложение B.1. Структура проекта

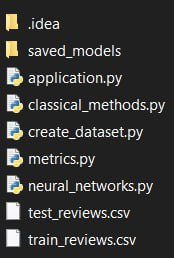
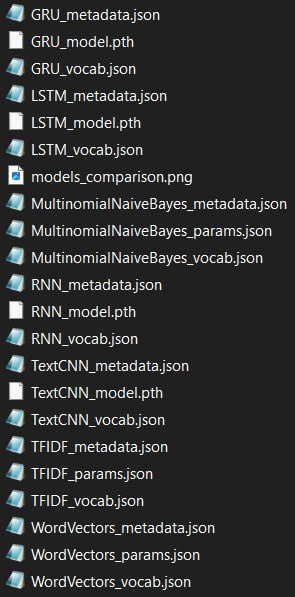


Рис.2: Файлы директории saved\_models

Рис.1: Файлы проекта

Приложение B.2. Результаты работы нейросетей

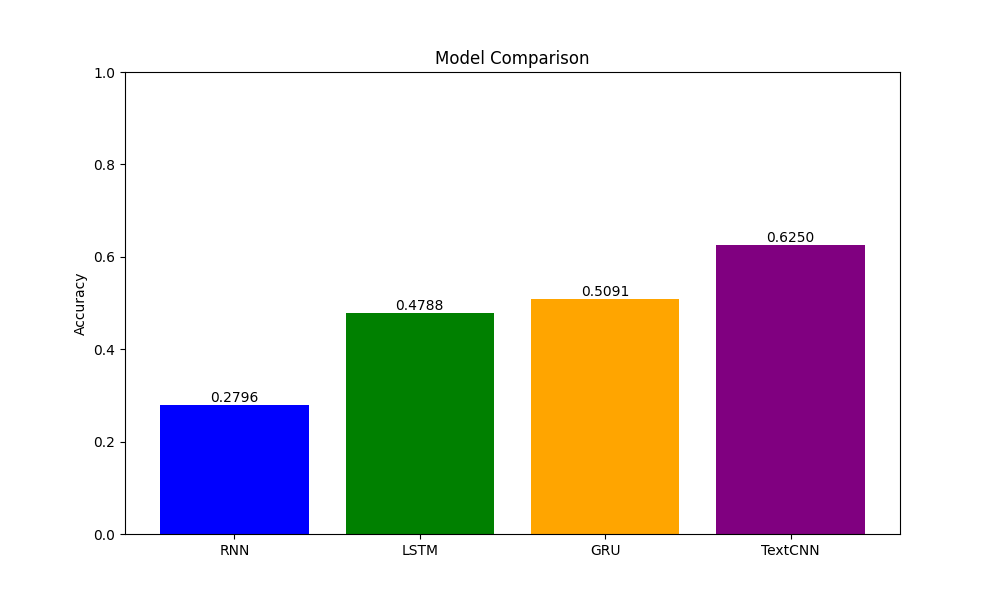


Рис.3: Сопоставление результатов работы нейронных моделей

Приложение B.3. Работа программы

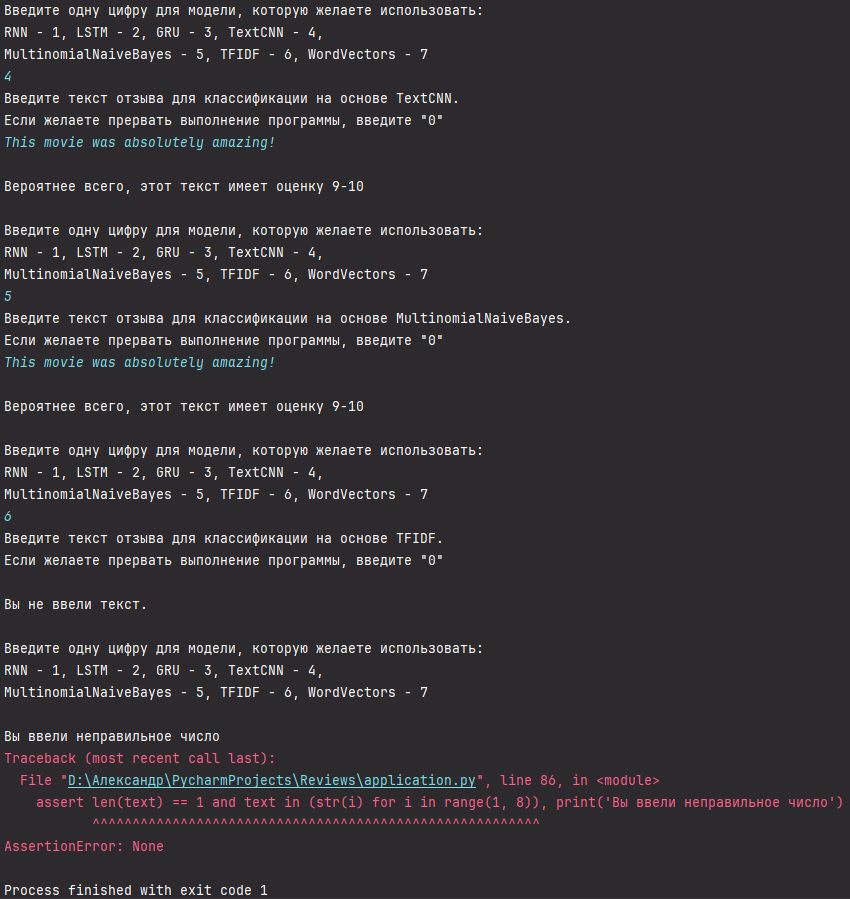


Рис.4: Пример использования программы

1. https://docs.python.org/3/library/collections.html [↑](#footnote-ref-1)
2. https://docs.python.org/3/library/csv.html [↑](#footnote-ref-2)
3. https://docs.python.org/3/library/json.html [↑](#footnote-ref-3)
4. https://docs.python.org/3/library/time.html [↑](#footnote-ref-4)
5. https://radimrehurek.com/gensim/ [↑](#footnote-ref-5)
6. https://matplotlib.org/ [↑](#footnote-ref-6)
7. https://www.nltk.org/ [↑](#footnote-ref-7)
8. https://numpy.org/ [↑](#footnote-ref-8)
9. https://pandas.pydata.org/ [↑](#footnote-ref-9)
10. https://www.selenium.dev/ [↑](#footnote-ref-10)
11. https://scikit-learn.org/stable/ [↑](#footnote-ref-11)
12. https://pytorch.org/ [↑](#footnote-ref-12)
13. https://imdb.com [↑](#footnote-ref-13)
14. https://www.imdb.com/list/ls063837343/ [↑](#footnote-ref-14)
15. https://www.imdb.com/chart/toptv/ [↑](#footnote-ref-15)
16. https://www.imdb.com/chart/bottom/ [↑](#footnote-ref-16)