Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«САРАТОВСКИЙ национальный исследовательский  
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ**

**Н. Г. ЧЕРНЫШЕВСКОГО»**

Факультет компьютерных наук и информационных технологий

УТВЕРЖДАЮ

Зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(ФИО)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата)

**ОТЧЕТ О ПРАКТИКЕ**

студентки 3 курса, 341 группы, факультета КНиИТ

Акишиной Полины Дмитриевны

(ФИО полностью, в родительном падеже)

вид практики: учебная ("Научно-исследовательская работа")

кафедра: кафедра информатики и программирования

(наименование кафедры)

курс: 3 .

(номер курса)

семестр: 5 .

(номер семестра)

продолжительность: 02.09.2024 – 27.12.2024

Руководитель практики:

зав. кафедрой \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Огнева М. В. .

(должность) (подпись, дата) (ФИО)

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 3](#_Toc184564484)

[1 Теоретическая часть 5](#_Toc184564485)

[1.1 Обзор источников 5](#_Toc184564486)

[1.2 Методы предобработки текстов для обучения нейронных сетей 7](#_Toc184564487)

[1.2.1 Очистка и форматирование текста 7](#_Toc184564488)

[1.2.2 Лемматизация и стемминг 8](#_Toc184564489)

[1.2.3 Векторизация текста 9](#_Toc184564490)

[1.2.4 Разделение данных на обучающую и тестовую выборки 11](#_Toc184564491)

[1.3 Метрики и методы оценки качества классификации текстов 11](#_Toc184564492)

[1.4 Выбор и описание архитектуры нейронной сети 13](#_Toc184564493)

[1.4.1 Входной и сверточный слои 13](#_Toc184564494)

[1.4.2 Слой объединения 14](#_Toc184564495)

[1.4.3 Полносвязный слой 15](#_Toc184564496)

[1.5 Методы оптимизации и предотвращения переобучения 16](#_Toc184564497)

[2 Используемые инструменты и библиотеки 18](#_Toc184564498)

[2.1 Инструменты для предобработки текстов 18](#_Toc184564499)

[2.2 Применение метрик качества 21](#_Toc184564500)

[2.3 Реализация модели нейронной сети 22](#_Toc184564501)

[ЗАКЛЮЧЕНИЕ 26](#_Toc184564502)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 27](#_Toc184564503)

# **ВВЕДЕНИЕ**

Автоматизация процессов в академической и исследовательской деятельности набирает популярность. С увеличением объема информации, особенно научных статей, важным становится автоматическое распределение докладов или публикаций по секциям конференций. Это снижает нагрузку на оргкомитеты и позволяет ускорить процесс организации. Кроме того, такая технология может стать полезным инструментом для оценки содержания статей, что улучшит систему публикаций. [1]

Нейронные сети и машинное обучение позволяют классифицировать тексты с высокой точностью, и это открывает перспективы для их использования в разных областях, включая автоматизацию научной деятельности. Разработка такой системы может существенно сократить время на обработку и распределение докладов, повысить объективность и качество принятия решений.

Целью курсовой работы является исследование возможности применения машинного обучения для автоматической обработки и классификации докладов, включая разработку и реализацию собственной классификации, а также оценка точности такой классификации в сравнении с традиционными методами. В соответствии с целью были выделены следующие задачи:

1. Изучить теоретические основы автоматической классификации текстов и нейронных сетей для обработки естественного языка .
2. Проанализировать существующие методы и подходы к автоматической классификации текстов.
3. Найти и собрать наборы данных научных статей или докладов, которые были ранее распределены по секциям.
4. Предобработать данные: очистить, нормализовать текст, подготовить к обучению модели.
5. Подобрать архитектуру нейронной сети для классификации текста.
6. Обучить нейронную сеть на размеченных данных для распознавания секций на основе текста.
7. Протестировать модель на новых данных, оценить её точность.
8. Сравнить результаты автоматической классификации с фактическими результатами, распределёнными вручную и оценить качество классификации с использованием метрик.
9. Провести анализ ошибок и выявить причины неверной классификации.
10. Рассмотреть возможности улучшения модели, в том числе через использование дополнительных данных или методов.

Задачами НИР пятого семестра являются задачи 1 – 5.

# **Теоретическая часть**

## **Обзор источников**

Для разработки нейронной сети, способной автоматически классифицировать научные статьи, необходимо изучить современные методы и подходы, используемые в обработке текстов и классификации данных. В последние годы в этой области произошёл значительный прогресс благодаря развитию методов обработки естественного языка (NLP) и глубокого обучения. Ниже представлен обзор ключевых направлений и подходов, используемых для решения задач классификации текстов.

Традиционные подходы к классификации текстов основываются на преобразовании текста в числовое представление с последующим применением алгоритмов машинного обучения. Среди классических методов выделяются:

* Наивный Байес. Метод основан на теореме Байеса и предполагает независимость признаков. Несмотря на свою простоту, он часто используется благодаря эффективности на малых данных. Он показал хорошие результаты при классификации спам-писем и анализа отзывов. Например, в статье [2] отмечается, что этот метод хорошо подходит для задач, где зависимости между словами минимальны. Однако этот подход имеет ограничение: он не учитывает порядок слов и их взаимосвязь.
* Метод k-ближайших соседей (k-NN). Этот алгоритм классифицирует объекты на основе их близости к уже размеченным данным. Он не требует обучения модели, что делает его удобным для небольших задач, но ресурсоёмким для больших объёмов данных. В обзоре [3] подчеркивается, что k-NN уступает по скорости более современным методам, таким как нейронные сети.
* Метод дерева решений. Деревья решений часто используются благодаря их интерпретируемости и способности учитывать нелинейные зависимости. Исследование, проведённое на корпусе русскоязычных текстов, показало, что метод дерева решений уступает более современным алгоритмам в точности, особенно при увеличении сложности текстов. [4]
* Метод опорных векторов (SVM). Метод SVM показывает хорошие результаты в задачах классификации текстов благодаря возможности работы с нелинейными границами. Он особенно эффективен для разреженных данных, полученных из текстов с использованием TF-IDF. Однако этот метод может быть вычислительно дорогим при работе с большими текстовыми корпусами. В обзоре [5] отмечается, что SVM эффективно справляется с задачами бинарной классификации, но уступает CNN в многоклассовых задачах.

Классические методы эффективны на небольших наборах данных и просты в реализации, однако они не учитывают контекст и последовательность слов, что ограничивает их точность при работе с большими текстами.

Современные подходы к обработке текстов всё чаще опираются на глубокие нейронные сети, которые способны учитывать сложные зависимости в данных и извлекать контекстуальные признаки.

* Рекуррентные нейронные сети (RNN): используются для последовательной обработки данных, что делает их полезными при работе с текстами. Однако они часто требуют больше времени на обучение. В статье [3] подчеркивается, что RNN уступают CNN в задачах обработки текстов средней длины из-за высокой вычислительной сложности.
* LSTM (Long Short-Term Memory) и GRU (Gated Recurrent Unit): это улучшенные версии RNN, которые помогают лучше захватывать долгосрочные зависимости в тексте. Эти модели широко используются для задач, связанных с анализом текста, таких как классификация, генерация и перевод текстов. Рекуррентные сети хорошо справляются с последовательными данными, но их обучение может быть медленным, особенно на больших наборах данных. [6]
* Свёрточные нейронные сети (CNN): изначально разработанные для обработки изображений, CNN также показали свою эффективность в задачах классификации текстов. Они могут захватывать локальные зависимости между словами и эффективно обрабатывать текстовые фрагменты. Статья [7] подчёркивает, что CNN превосходят традиционные методы, такие как SVM и наивный Байес, в задачах тематической классификации текстов. Это обусловлено их способностью анализировать слова в контексте их окружения, что особенно важно для научных текстов.

Современные нейронные сети являются ведущими решениями для автоматической классификации текстов. Однако остаются вопросы относительно оптимизации этих моделей для конкретных задач и использования вычислительных ресурсов, что требует дальнейшего исследования.

Проведённый обзор показывает, что традиционные подходы, несмотря на простоту, не способны справляться со сложными текстами и большими корпусами данных. Современные методы, такие как CNN, предоставляют значительные преимущества, включая способность учитывать контекст и взаимосвязь слов. Исследования подтверждают, что CNN лучше подходят для классификации научных текстов, обеспечивая высокую точность при сравнительно низких требованиях к ресурсам

## **Методы предобработки текстов для обучения нейронных сетей**

Предобработка текстовых данных — это важнейший этап в разработке нейронной сети для автоматической классификации научных статей. От качества и глубины предобработки напрямую зависит, насколько эффективно модель сможет распознавать ключевые структуры и классифицировать тексты. Каждый шаг в предобработке нацелен на то, чтобы сделать данные более понятными для модели, устранив ненужный шум и оптимизировав подачу информации. Рассмотрим каждый этап предобработки данных с примерами и указанием практических шагов.

### **Очистка и форматирование текста**

Первый шаг в предобработке — это удаление лишней информации, которая не несет значимой смысловой нагрузки для задачи классификации. Сюда включаются:

* Знаки пунктуации (точки, запятые, кавычки).
* Специальные символы (например, %, $, @).
* Числа, если они не имеют значения для конкретного корпуса текстов.

В ряде исследований подчеркивается, что очистка текста существенно влияет на точность моделей классификации, так как избавляет модель от лишних данных, которые могут исказить результат (например, как в работе, посвященной классификации текстов с помощью сверточных нейронных сетей). [8]

В текстах научных статей могут встречаться цифры для обозначения номеров разделов, номеров таблиц или ссылок на рисунки (например, «Рис. 1», «Таблица 3.2»). Эти данные могут не иметь значения для классификации текста по темам, поэтому их нужно удалить. Текст «В Таблице 3 представлены результаты эксперимента» превращается в «В представлены результаты эксперимента».

Приведение текста к единому регистру позволяет избежать ошибок, связанных с различием между заглавными и строчными буквами. В русском языке, например, слова «модель» и «Модель» имеют одинаковое значение, но без приведения к нижнему регистру они будут восприниматься моделью как разные. На практике это делается через простую функцию преобразования строки в нижний регистр. В Python это метод **lower()**.

### **Лемматизация и стемминг**

**Лемматизация — это процесс приведения слова к его начальной форме, называемой леммой. Лемма — это базовая, каноническая форма слова, которая представлена в словаре (например, для глаголов — это инфинитив, для существительных — именительный падеж). Лемматизация учитывает грамматические особенности языка и возвращает исходное слово к форме, которая сохраняет смысл, но убирает склонения, спряжения и прочие морфологические изменения. [9]**

**Примеры лемматизации: «бегу» — «бег», «сильный» — «сила», «читал» — «читать».**

**Стемминг — это процесс уменьшения слова до его основы (стемы) путем удаления суффиксов, префиксов или окончаний. В отличие от лемматизации, стемминг не обращает внимания на грамматические особенности, что делает его более простым, но иногда менее точным. Он не всегда возвращает реальное слово, а просто сокращает его до стема.**

Примеры стемминга: «бегу», «бежал», «бегущий» **—** «бег»; «читал», «читать», «читающий» **—** «чит»; «модели», «моделировать» **—** «модел».

Применение этих методов позволяет уменьшить количество уникальных слов в корпусе данных, что облегчает задачу нейронной сети. Исследования показывают, что лемматизация чаще всего даёт лучшие результаты для сложных текстов, таких как научные статьи. [7]

**Стемминг** полезен для более быстрых операций, но он может исказить некоторые слова, что недопустимо для научных текстов, по этой причине он не будет использоваться в данной работе.

### **Векторизация текста**

После очистки и нормализации текста необходимо преобразовать его в числовое представление, чтобы нейронная сеть могла работать с ним. Для этого используются такие методы векторизации текста, как Мешок слов, TF-IDF, Векторное представление слов

**Мешок слов (Bag of Words, BoW)** — это один из самых простых методов векторизации, где каждый документ представляется как набор слов, а вектор текста состоит из чисел, обозначающих количество вхождений каждого слова. При этом каждое уникальное слово в корпусе (наборе текстов) становится элементом вектора, а его значение в документе указывает, сколько раз это слово встречается. Итоговый вектор для текста фиксированной длины содержит количество вхождений всех слов из словаря корпуса. Это приводит к большим, разреженным векторам, если корпус содержит много уникальных слов. [10]

BoW не учитывает порядок слов, поэтому он теряет информацию о контексте, что особенно важно для текстов, где порядок слов влияет на значение. Для больших текстовых массивов метод может быть ресурсоёмким из-за большого количества уникальных слов и разреженности вектора.

**TF-IDF (Term Frequency - Inverse Document Frequency) – у**лучшенная версия BoW, она вводит весовые коэффициенты для слов, которые зависят не только от их частоты в тексте, но и от частоты встречаемости в других документах. TF-IDF рассчитывается по формуле:



где:

* TF (Term Frequency) — частота слова t в документе d.
* IDF (Inverse Document Frequency) — обратная частота документа, показывающая, насколько редким является слово в корпусе.

Слова, которые встречаются в большом количестве документов, получают меньший вес, что снижает их значимость. Этот метод делает вектор более информативным, так как он позволяет снизить вес общих слов (например, «и», «в», «на») и выделить более редкие и значимые слова, что делает его особенно полезным для задач, требующих анализа контента. [11]

TF-IDF также не учитывает порядок слов, из-за чего теряется часть семантической информации.

**Векторное представление слов (Word Embeddings)** — это более продвинутый метод векторизации, который представляет каждое слово как многомерный вектор, в котором кодируется семантическая информация. Одной из популярных моделей для создания Word Embeddings является Word2Vec. В отличие от BoW и TF-IDF, Word2Vec может учитывать контекст, что позволяет ему улавливать семантическую близость между словами. Word2Vec работает по двум основным подходам:

* CBOW (Continuous Bag of Words): предсказывает целевое слово на основе контекста, т.е. слов, окружающих его.
* Skip-gram: предсказывает слова контекста для данного целевого слова.

Обучение Word2Vec приводит к тому, что слова с похожим значением оказываются рядом в многомерном пространстве. Например, слова «сеть» и «нейронные» окажутся близкими в пространстве, что улучшает понимание текста моделью. Векторы фиксированной длины, получаемые с помощью Word2Vec, содержат не частотную, а семантическую информацию, что позволяет эффективно учитывать близость значений слов в тексте. [10][11]

Word2Vec требует больших данных для качественного обучения и значительных вычислительных ресурсов.

### **Разделение данных на обучающую и тестовую выборки**

Для обучения модели важно разделить исходный набор данных на обучающую и тестовую выборки. Обучающая выборка используется для настройки модели, а тестовая — для оценки её качества на данных, которые модель не видела.

Если существует набор данных из 1000 статей, 800 из них могут использоваться для обучения модели, а оставшиеся 200 — для тестирования. Это стандартный подход, позволяющий избежать проблемы **переобучения**, когда модель слишком точно запоминает обучающие данные, но плохо обобщает на новых.

В Python библиотека **scikit-learn** предоставляет удобный метод **train\_test\_split** для разделения данных.

Таким образом, предобработка данных является важнейшим шагом, от которого напрямую зависит эффективность последующего обучения нейронной сети. Очистка текста, нормализация, а также выбор подходящего метода векторизации играют ключевую роль в процессе автоматической классификации научных статей.

## **Метрики и методы оценки качества классификации текстов**

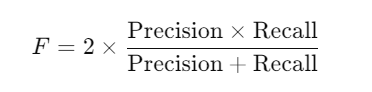
Для оценки эффективности моделей классификации текстов в задачах машинного обучения применяются специальные метрики, которые позволяют оценить качество предсказаний, сделанных моделью. В этой главе рассмотрены основные метрики, такие как **точность (Precision)**, **полнота (Recall)**, **F-мера**, **Accuracy** и **AUC-ROC**. Эти метрики используются для оценки моделей, работающих с различными типами текстов, включая научные тексты.

**Precision** показывает, какой процент от всех объектов, отнесённых моделью к определённому классу, действительно принадлежит этому классу. Высокая точность означает, что модель делает мало ложных положительных предсказаний. Применение этой метрики особенно важно, когда неверная классификация может привести к нежелательным последствиям, например, если научные статьи разделяются по ключевым темам для последующего анализа​. [12]

**Recall** измеряет, какую долю от всех объектов данного класса модель правильно идентифицировала как принадлежащие этому классу. В задачах, где важно максимально распознавать все элементы определённой категории, высокая полнота имеет решающее значение​. [4]

**F-мера** представляет собой гармоническое среднее между точностью и полнотой, предоставляя сбалансированную оценку качества модели. Она особенно полезна в ситуациях, когда требуется учитывать как ложноположительные, так и ложноотрицательные ошибки. F-мера часто применяется в задачах, связанных с анализом художественных текстов или научных публикаций, где необходимо одновременно учитывать точность и полноту классификации. [13]

Формула для вычисления F-меры:

****

**Accuracy** (доля правильных предсказаний) — это одна из простейших метрик, которая показывает, какой процент предсказаний модель сделала правильно по отношению ко всем предсказаниям. Однако при использовании этой метрики важно учитывать баланс классов, так как в случае значительного перекоса Accuracy может быть неинформативной.

**AUC-ROC** измеряет способность модели различать классы на основе вероятностей. Она используется для визуализации и оценки качества бинарных классификаторов, показывая, насколько хорошо модель отделяет положительные примеры от отрицательных. В случае задач классификации текстов эта метрика может показать, насколько уверенно модель распознаёт разные категории, что особенно полезно при классификации научных статей по тематикам​. [9]

Для комплексной оценки качества моделей классификации текстов в задачах машинного обучения рекомендуется использовать сочетание метрик — Precision, Recall и F-меры, чтобы учесть различные аспекты качества предсказаний. При этом AUC-ROC может быть полезна для визуализации результатов на тестовой выборке и для задач с несбалансированными данными, где важен учёт вероятностей.

## **Выбор и описание архитектуры нейронной сети**

Для решения задачи классификации научных текстов на основе анализа их содержания была выбрана свёрточная нейронная сеть (CNN). Свёрточные нейронные сети лучше других архитектур справляются с выделением локальных признаков в текстах. Они способны эффективно обучаться на больших массивах данных, выделяя ключевые слова и фразы, характерные для каждой темы, и игнорируя незначительные детали. Это особенно важно для научных текстов, где терминология и сочетания слов могут указывать на определённую предметную область. Кроме того, в отличие от рекуррентных нейронных сетей, которые обрабатывают текст последовательно и требуют больше ресурсов для обучения, CNN позволяют проводить обработку параллельно, что ускоряет обучение и повышает производительность​. Свёрточные нейронные сети способны выделять иерархические признаки, что делает их особенно подходящими для текстов средней и большой длины, где важны как локальные зависимости (например, последовательности терминов), так и семантические особенности текста в целом. [7][14]

Рассмотрим более подробно архитектуру выбранной нейронной сети.

### **Входной и сверточный слои**

Входной слой получает текст в виде числовых данных. Для этого предварительно используется метод векторизации, в данном случае, TF-IDF. Метод помогает отфильтровывать часто встречающиеся слова и выделять специфические, редкие термины, которые могут указывать на тематику текста. Результат будет содержать числовые представления для каждого текста в корпусе, где наиболее значимые слова (на основе частоты их встречаемости) имеют более высокие значения. Эта матрица затем подаётся на вход свёрточной нейронной сети. [15]

Свёрточные слои (CNN) играют ключевую роль в анализе текстов, так как позволяют выделить характерные признаки, такие как словосочетания и фразы, указывающие на тематику или тональность текста. В свёрточных нейронных сетях используются фильтры (ядра), которые проходят по матрице признаков (в нашем случае, по TF-IDF векторах), захватывая группы слов. Это позволяет выделить локальные паттерны, которые могут быть полезными для классификации научных статей по темам. Для текстовой классификации часто используют фильтры размером 2 или 3, что позволяет захватывать пары (биграммы) или тройки слов (триграммы). [16]

После того как фильтры применяются к тексту, они выделяют набор признаков — чисел, которые отражают активность определённых комбинаций слов. Эти значения показывают, насколько хорошо данный фрагмент текста соответствует искомому признаку, например, тематике.

Каждый фильтр изучает текст на различных уровнях и применяет себя ко всем возможным комбинациям слов в предложении. Это создает карту признаков (feature map), отражающую выделенные паттерны для каждого текста.

### **Слой объединения**

Слой объединения или pooling layer используется в нейронных сетях для уменьшения размерности данных, при этом сохраняя наиболее важные признаки. В задачах классификации текста наиболее распространённым подходом является max-pooling — метод, который извлекает максимальные значения из каждой группы элементов, полученных из свёрточного слоя. Это позволяет сохранить наиболее значимые признаки, игнорируя менее важные, что помогает сократить размер данных и сделать модель более устойчивой к незначительным вариациям в тексте.

В слое max-pooling определяется размер окна, например 2 или 3. Это окно перемещается по выходной матрице признаков свёрточного слоя, захватывая значения в каждой его области. Размер окна определяет, сколько соседних значений будут сгруппированы для извлечения максимального. [14]

Для каждого положения окна выбирается максимальное значение из захваченных элементов, которое и становится значением для текущей позиции на выходе pooling-слоя. Это уменьшает размер матрицы признаков, оставляя только наибольшие активации, то есть наиболее значимые признаки.

### **Полносвязный слой**

После объединяющих слоёв используется полносвязный слой, который завершает архитектуру нейронной сети, объединяя и обобщая признаки, выделенные свёрточными и pooling-слоями. Этот слой играет ключевую роль в классификации, так как именно здесь происходит окончательная обработка признаков для предсказания категории текста.

Полносвязный слой принимает на вход все признаки, выделенные предыдущими слоями, и преобразует их в одномерный вектор. Каждый нейрон в этом слое соединён с каждым входным значением, что позволяет слою учитывать всю доступную информацию. [7]

Перед подачей данных на полносвязный слой их необходимо «выпрямить» — превратить многомерный массив признаков в одномерный. Это упрощает процесс классификации и позволяет соединить выходы свёрточных и pooling-слоёв.

Выходной слой использует функцию активации softmax, которая преобразует выходные значения модели в вероятности для каждого класса. Таким образом, на выходе получается вероятность принадлежности текста к каждому из возможных классов, и тексту присваивается класс с наибольшей вероятностью. [17]

Так, для задачи классификации с тремя классами (например, статьи по "математике", "информатике" и "физике") выход softmax может быть, например, [0.1, 0.7, 0.2], что говорит о 70% вероятности принадлежности текста к "информатике".

## **Методы оптимизации и предотвращения переобучения**

В задачах машинного обучения переобучение (overfitting) возникает, когда модель слишком точно запоминает обучающие данные, но плохо обобщает на новые, невидимые данные. Это часто случается с нейронными сетями, особенно при небольшом объёме данных. В этом пункте рассмотрим несколько техник для предотвращения переобучения, включая Dropout, L2-регуляризацию и Early Stopping. Эти методы оптимизации позволяют улучшить способность модели обобщать и предсказывать на тестовой выборке. [19]

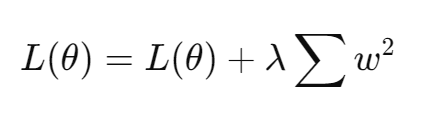
Dropout — это техника регуляризации, специально разработанная для предотвращения переобучения, которая случайным образом отключает определённый процент нейронов на каждом этапе обучения. Это препятствует тому, чтобы нейроны становились излишне зависимыми друг от друга, и улучшает способность модели обобщать информацию на новых данных.

При каждом шаге обучения случайный процент нейронов, выбранный в зависимости от гиперпараметра (например, 0.3 или 0.5), временно отключается. В результате модель не может полагаться на конкретные нейроны и вынуждена искать более устойчивые связи в данных, улучшая таким образом свою устойчивость к переобучению. [20]

L2-регуляризация (также называемая регуляризацией весов или сглаживанием весов) добавляет штраф к значению функции потерь, зависящий от величины весов модели. Это побуждает модель искать решение с минимальными значениями весов, что делает её менее склонной к запоминанию деталей обучающих данных.

L2-регуляризация добавляет к функции потерь дополнительный член, пропорциональный квадрату значений всех весов сети. Это заставляет модель минимизировать не только ошибку, но и величину весов, помогая ей избегать излишней зависимости от отдельных нейронов.

Формула функции потерь с L2-регуляризацией:



где:

* L(θ) — исходная функция потерь,
* ∑w2 — сумма квадратов всех весов,
* λ — коэффициент регуляризации. [20]

Early Stopping — это метод, который завершает процесс обучения. Когда модель начинает переобучаться, точность на обучающей выборке продолжает расти, а на валидационной — начинает снижаться. Early Stopping автоматически завершает обучение в этот момент, предотвращая переобучение. [21]

Early Stopping отслеживает метрику (например, валидационную ошибку) на каждом этапе обучения. Если метрика перестаёт улучшаться на протяжении нескольких эпох, обучение завершается, предотвращая переобучение и сокращая время обучения.

# **Используемые инструменты и библиотеки**

В этом пункте будут представлены основные инструменты и библиотеки Python, которые используются для предобработки текста, обучения модели и оценки её качества. Эти методы помогают стандартизировать тексты, преобразовывать их в числовое представление и корректно делить на выборки для обучения и тестирования.

## **Инструменты для предобработки текстов**

Для начального этапа очистки текста, включая удаление чисел, знаков пунктуации и других нежелательных символов, в Python применяется встроенная библиотека **re** для регулярных выражений и модуль **string**. Эти методы помогают подготовить текст к дальнейшей обработке, удаляя символы и числа, которые могут искажать результаты анализа.

import re  
import string  
  
*# Пример удаления чисел и пунктуации*def clean\_text(text):  
 text = re.sub(r'\d+', '', text) *# Удаляем цифры* text = text.translate(str.maketrans('', '', string.punctuation)) *# Удаляем пунктуацию* return text.lower()

Приведение текста к одному регистру выполняется с помощью метода lower(), встроенного в Python. Это простой, но важный шаг для того, чтобы избежать различий между словами, написанными с заглавной или строчной буквы.

text = "Нейронные Сети используются в Машинном обучении"  
cleaned\_text = text.lower()  
print(cleaned\_text) *# "нейронные сети используются в машинном обучении"*

Для лемматизации в Python используется библиотека **pymorphy2**, которая способна точно лемматизировать слова, учитывая грамматику и контекст русского языка. Лемматизация применяется для приведения слов к их базовой форме, что важно для работы с научными текстами, где точность имеет особое значение.

import pymorphy2  
  
morph = pymorphy2.MorphAnalyzer()  
  
def lemmatize\_text(text):  
 words = text.split()  
 lemmatized = [morph.parse(word)[0].normal\_form for word in words]  
 return " ".join(lemmatized)  
  
text = "Модели, использующие нейронные сети, активно развиваются."  
lemmatized\_text = lemmatize\_text(text)  
print(lemmatized\_text) *# "модель использовать нейронный сеть активно развиваться"*

Для задач, где важна скорость обработки, можно использовать **стемминг** (например, с библиотекой **nltk** и алгоритмом PorterStemmer), но с риском потери грамматической точности.

from nltk.stem import PorterStemmer  
from nltk.tokenize import word\_tokenize  
  
stemmer = PorterStemmer()  
text = "running runners runs easily fairly"  
words = word\_tokenize(text)  
stemmed\_words = [stemmer.stem(word) for word in words]  
print(stemmed\_words)

Для векторизации текста используются такие популярные инструменты, как scikit-learn и gensim.

TF-IDF (Term Frequency-Inverse Document Frequency): этот метод реализуется с помощью TfidfVectorizer из библиотеки scikit-learn. Он взвешивает частоту слов, придавая больший вес редким и информативным словам и меньший — часто встречающимся словам, что помогает улучшить качество классификации.

from sklearn.feature\_extraction.text import TfidfVectorizer  
  
corpus = [  
 "нейронные сети используются для классификации текстов",  
 "машинное обучение активно развивается",  
 "классификация текстов с использованием нейронных сетей"  
]  
  
vectorizer = TfidfVectorizer()  
X = vectorizer.fit\_transform(corpus)  
  
print(vectorizer.get\_feature\_names\_out())  
print(X.toarray())

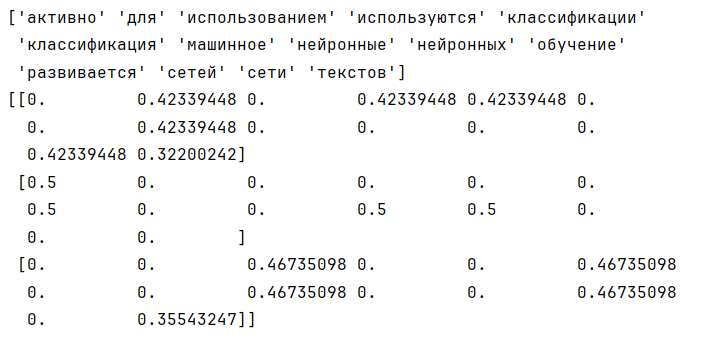


Рисунок 1 - Результат работы программы для TF-IDF

**Word2Vec**: этот метод доступен в **gensim** и используется для представления слов в виде плотных векторов, которые отражают семантические связи. Векторы обучаются на текстовом корпусе, а затем каждый вектор используется для представления значения слова. Word2Vec полезен для задач, требующих учета контекста.

import gensim  
from gensim.models import Word2Vec  
import re  
  
*# Пример корпуса текстов (каждая статья разбита на предложения)*corpus = [  
 "Нейронные сети активно используются для классификации текста.",  
 "Машинное обучение развивается быстрыми темпами.",  
 "Классификация текста является одной из задач машинного обучения.",  
 "Нейронные сети применяются в различных областях искусственного интеллекта."  
]  
  
*# Очистка текста и токенизация*def preprocess\_text(text):  
 text = re.sub(r'\W', ' ', text) *# Удаление знаков пунктуации* text = text.lower() *# Приведение к нижнему регистру* text = text.split() *# Токенизация (разбиение текста на слова)* return text  
  
*# Подготовка данных*processed\_corpus = [preprocess\_text(doc) for doc in corpus]  
  
*# Обучение модели Word2Vec  
# Используем CBOW (по умолчанию) для обучения*model = Word2Vec(sentences=processed\_corpus, vector\_size=100, window=5, min\_count=1, workers=4)  
  
*# Пример получения вектора для слова*word = "нейронные"  
vector = model.wv[word] *# Вектор для слова "нейронные"*print(f"Вектор для слова '{word}':\n{vector}")

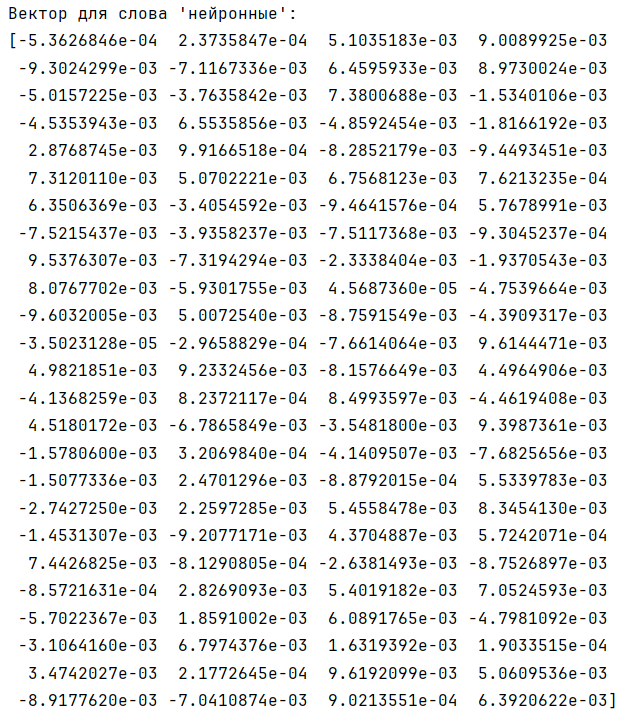


Рисунок 2 - Результат работы программы для Word2Vec

Для разделения данных на обучающую и тестовую выборки применяется метод train\_test\_split из библиотеки **scikit-learn**. Это помогает избежать проблемы переобучения, позволяя модели проверяться на данных, которые она не видела во время обучения.

from sklearn.model\_selection import train\_test\_split  
  
X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

Эти инструменты и библиотеки позволяют провести все необходимые этапы предобработки данных, векторизации и подготовки выборок для обучения нейронной сети.

## **Применение метрик качества**

Для практической оценки качества модели классификации текстов в Python можно использовать библиотеку **scikit-learn**, которая предоставляет функции для расчёта таких метрик, как точность (Precision), полнота (Recall), F-мера, Accuracy и AUC-ROC. Рассмотрим, как эти метрики могут быть применены к обученной модели.

Предположим, что у нас есть обученная модель и предсказания для тестового набора данных. Метки истинных классов хранятся в переменной y\_test, а предсказания модели — в y\_pred.

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score, roc\_auc\_score  
  
*# Предполагаем, что y\_test - истинные метки классов, а y\_pred - предсказанные моделью метки  
  
# Accuracy*accuracy = accuracy\_score(y\_test, y\_pred)  
print(f"Accuracy: {accuracy}")  
  
*# Precision*precision = precision\_score(y\_test, y\_pred, average='weighted') *# 'weighted' для учета дисбаланса классов*print(f"Precision: {precision}")  
  
*# Recall*recall = recall\_score(y\_test, y\_pred, average='weighted')  
print(f"Recall: {recall}")  
  
*# F-мера*f1 = f1\_score(y\_test, y\_pred, average='weighted')  
print(f"F-мера: {f1}")  
  
*# AUC-ROC (для вероятностей предсказания)  
# Если модель выдаёт вероятности, используем их для расчёта AUC-ROC  
# Предполагаем, что y\_pred\_prob - вероятности для положительного класса*y\_pred\_prob = model.predict\_proba(X\_test)[:, 1] *# вероятности для положительного класса*auc\_roc = roc\_auc\_score(y\_test, y\_pred\_prob)  
print(f"AUC-ROC: {auc\_roc}")

**Precision и Recall р**ассчитываются с использованием параметра average='weighted' для учёта возможного дисбаланса классов. Это важно, если одна категория встречается реже других. Если классов больше двух, используется average='macro', что даёт усреднённые значения по каждому классу.

## **Реализация модели нейронной сети**

Свёрточные слои служат для обработки матрицы признаков, созданной с помощью TF-IDF. В каждом слое применяются фильтры (обычно размером 3 или 5), которые скользят по матрице, извлекая локальные паттерны из фраз и словосочетаний. Для реализации свёрточных слоёв можно использовать библиотеку Keras с функцией Conv1D, которая применяется к одномерным данным.

from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Conv1D, Dense, MaxPooling1D, Flatten  
import numpy as np  
  
*# Создаем модель*model = Sequential()  
*# Добавляем свёрточный слой*model.add(Conv1D(filters=64, kernel\_size=3, activation='relu', input\_shape=(tfidf\_matrix.shape[1], 1)))  
  
*# Вывод структуры модели*model.summary()

В этом примере используется фильтр размером 3, который извлекает трёхсловные последовательности в тексте, что подходит для извлечения фраз, часто встречающихся в научных статьях.

Max-pooling слои помогают уменьшить размер данных и выделить наибольшие значения, представляющие наиболее активные признаки. Max-pooling улучшает устойчивость модели к незначительным вариациям и предотвращает переобучение. В Keras слой MaxPooling1D добавляется после свёрточного слоя, чтобы уменьшить размер данных.

*# Добавляем слой max-pooling*model.add(MaxPooling1D(pool\_size=2))

Здесь pool\_size=2 означает, что слой объединяет два соседних значения, оставляя максимальное значение, которое лучше всего отражает важные признаки.

Полносвязный слой позволяет комбинировать признаки, выделенные на предыдущих этапах, и классифицировать текст. Полносвязные слои объединяют всю информацию и делают финальное предсказание.

*# Добавляем слой Flatten для выравнивания данных перед полносвязным слоем*model.add(Flatten())  
*# Добавляем полносвязный слой*model.add(Dense(128, activation='relu'))

В этом примере используется слой Flatten, чтобы преобразовать данные для подачи в полносвязный слой, а затем Dense с 128 нейронами, который помогает модели обучиться на основе высокоуровневых признаков текста.

Функция активации softmax применяется в выходном слое, преобразуя данные в вероятности принадлежности к каждому классу. Это особенно важно для многоклассовой классификации, так как softmax позволяет модели предсказывать вероятность принадлежности текста к каждому возможному классу.

*# Добавляем выходной слой с softmax для многоклассовой классификации*model.add(Dense(3, activation='softmax')) *# Предположим, у нас 3 класса*

Этот слой предсказывает вероятность для каждого из трёх классов. Например, для задачи классификации научных статей по темам он может вернуть вероятности принадлежности текста к "информатике", "математике" или "физике".

В этом же пункте рассмотрим реализацию методов для предотвращения переобучения.

Использование метода регуляризации Dropout на каждом слое позволяет существенно снизить риск переобучения. В Python Dropout, как правило реализуется с помощью библиотеки Keras:

from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense, Dropout  
  
*# Инициализация модели*model = Sequential()  
model.add(Dense(128, activation='relu', input\_shape=(input\_dim,)))  
model.add(Dropout(0.5)) *# 50% нейронов будут отключены*model.add(Dense(64, activation='relu'))  
model.add(Dropout(0.3)) *# 30% нейронов будут отключены*model.add(Dense(3, activation='softmax'))

В данном примере Dropout(0.5) означает, что на каждом шаге обучения половина нейронов будет случайным образом отключена.

В Keras также можно добавить L2-регуляризацию с помощью параметра kernel\_regularizer:

from tensorflow.keras.regularizers import l2  
from tensorflow.keras.models import Sequential  
from tensorflow.keras.layers import Dense  
  
model = Sequential()  
model.add(Dense(128, activation='relu', kernel\_regularizer=l2(0.01), input\_shape=(input\_dim,)))  
model.add(Dense(64, activation='relu', kernel\_regularizer=l2(0.01)))  
model.add(Dense(3, activation='softmax'))

Значение kernel\_regularizer=l2(0.01) добавляет L2-регуляризацию с коэффициентом 0.01. Это значение подбирается экспериментально — слишком высокое значение может привести к недообучению, а слишком низкое — к недостаточной регуляризации.

Early Stopping можно реализовать с помощью обратного вызова EarlyStopping:

from tensorflow.keras.callbacks import EarlyStopping  
  
early\_stopping = EarlyStopping(monitor='val\_loss', patience=5, restore\_best\_weights=True)  
  
model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])  
history = model.fit(X\_train, y\_train, epochs=50, validation\_split=0.2, callbacks=[early\_stopping])

Здесь monitor='val\_loss' указывает, что Early Stopping будет следить за ошибкой на валидационной выборке. Значение patience=5 задаёт количество эпох, в течение которых модель может не улучшаться перед завершением обучения. Параметр restore\_best\_weights=True восстановит веса модели, при которых она показала наилучшие результаты.

# **ЗАКЛЮЧЕНИЕ**

В соответствии с поставленными задачами, в ходе работы был проведен теоретический анализ методов автоматической классификации текстов с использованием нейронных сетей. Основное внимание было уделено изучению современных подходов обработки текстов, включая этапы предобработки, выбор методов векторизации, архитектуры нейронной сети и метрик для оценки качества классификации.

Проведенный обзор показал, что современные подходы, такие как сверточные нейронные сети, эффективно решают задачи текстовой классификации благодаря способности выделять локальные иерархические признаки текста. Кроме того, рассмотрены ключевые методы предотвращения переобучения, включая Dropout, L2-регуляризацию и раннюю остановку, что способствует улучшению обобщающей способности моделей.

Таким образом, результат исследования предоставляет основы для дальнейшей разработки и практической реализации модели классификации текстов с применением нейронной сети, включая оптимизацию ее архитектуры и оценку эффективности на реальных данных.

# **СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

1. Олейников, Б. В., Шалабай, А. И. Классификация научных текстов для реализации тематического поиска в специализированных системах. // Международный научно-исследовательский журнал. – 2013. – С. 103-105
2. Бирназаров, Р. Р. Литературный обзор современных методов классификации текстовых данных // Colloquium-journal. – 2024. - №9(202). – С. 32-35
3. Максютин, П. А. , Шульженко, С. Н. Обзор методов классификации текстов с помощью машинного обучения // Инженерный вестник Дона. – 2022. – №12
4. Челышев, Э. А., Оцоков, Ш. А., Раскатова, М. В., Щёголев, П. Сравнение методов классификации русскоязычных новостных текстов с использованием алгоритмов машинного обучения. // Вестник кибернетики. – 2022. – №1 (45)
5. Казанцев, А. А., Прохоров, М. В., Худякова, П. С. Обзор подходов к классификации текстов актуальными методами // Экономика и качество систем связи. – 2021. – №1 – С. 57-66
6. Будыльский, Д. В. GRU и LSTM: современные рекуррентные нейронные сети // Молодой ученый. — 2015. — № 15 (95). — С. 51-54.
7. Воробьев, Е. В., Пучков, Е. В. Классификация текстов с помощью сверточных нейронных сетей // Молодой исследователь Дона. – 2017.
8. Бобков С. П., Суворов, С. В., Фролов, И. А., Казадаев, А. И. Разработка классификационного алгоритма текстовой информации на основе обращений пользователей интернет-ресурсов с использованием нейронных сетей // Известия высших учебных заведений. Серия: Экономика, финансы и управление производством. – 2019. – №02(40)
9. Батура, Т.В. Методы автоматической классификации текстов // Программные продукты и системы. – 2017. – Т. 30 № 1. С. 85-99
10. Шевелев, О. Г., Петраков, А. В. Классификация текстов с помощью деревьев решений и нейронных сетей прямого распространения. // Вестник Томского государственного университета. – 2006.
11. Гальцева, Т. В., Нестеров, С. А. Классификация и определение тональности текстов, публикуемых в сети интернет. // Системный анализ в проектировании и управлении. – 2024.
12. Ермакова, Л. М. Методы классификации текстов и определения качества контента. // Вестник Пермского университета. Серия: Математика. Механика. Информатика. – 2011. – Вып. 3(7)
13. Ашуров, М. Ф. Сравнение потоковых методов классификации текстов художественной литературы на основе сжатия информации и подсчета подстрок. // Вестник Томского государственного университета. Управление, вычислительная техника и информатика. – 2014. – №4 (29)
14. Смирнова, О. С., Шишков, В. В. Выбор топологии нейронных сетей и их применение для классификации коротких текстов. // International Journal of Open Information Technologies. – 2016. – vol. 4, no. 8
15. Черкасов, А. Н., Туркин, Е. А. Выбор оптимальной архитектуры искусственной нейронной сети для задачи классификации текстов. // Вестник Адыгейского государственного университета. Серия 4: Естественно-математические и технические науки. – 2021. – выпуск 1 (276)
16. Трифонов, К. В. Сравнение сверточной и рекуррентной архитектур нейронных сетей при решении задачи анализа тональности текста. // Молодой исследователь Дона. – 2024. – 9(2):41-44
17. Глазкова, А. В. Сравнение нейросетевых моделей для классификации текстовых фрагментов, содержащих биографическую информацию. // Программные продукты и системы. – 2019. – Т. 32. № 2
18. Бредихин, А. И. Алгоритмы обучения сверточных нейронных сетей. // Вестник Югорского государственного университета. – 2019. – выпуск 1 (52)
19. Пичугин, Р. А. Методы борьбы с переобучением в нейронных сетях. // Международный журнал гуманитарных и естественных наук. – 2022. – vol. 7-2 (70)
20. Жуков, Л. А., Хомич, А. В. Эффект структурного переобучения при оптимизации структур нейронных сетей. // Новые информационные технологии в автоматизированных системах. – 2006.
21. Федосин, С. А., Ладяев, Д. А, Марьина, О. А. Анализ и сравнение методов обучения нейронных сетей. // Инженерные технологии и системы. – 2010.
22. Батура Т.В. Методы автоматической классификации текстов // Программные
23. продукты и системы. 2017. Т. 30. № 1. С. 85–99; DOI: 10.15827/0236-235X.030.1.085-
24. 099.
25. Батура Т.В. Методы автоматической классификации текстов // Программные
26. продукты и системы. 2017. Т. 30. № 1. С. 85–99; DOI: 10.15827/0236-235X.030.1.085-
27. 09