МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«МИРЭА – Российский технологический университет»
РТУ МИРЭА

РАБОТА ДОПУЩЕНА К ЗАЩИТЕ

Руководитель
программы _____ Ш.Г. Магомедов

«30» мая 2025 г.

ИТОГОВАЯ АТТЕСТАЦИОННАЯ РАБОТА

по дополнительной программы профессиональной переподготовки
«Программные средства решения прикладных задач искусственного интеллекта»

На тему: «Модель распознавания вариант 146»

Обучающийся _____ Гаджиханов Хаджимурат Русланович
Подпись *Фамилия, имя, отчество*

группа ИВБО-07-21

Руководитель работы _____ Ш.Г. Магомедов
подпись

Москва 2025 г.

Гаджиханов Хаджимурат Русланович
ИВБО-07-21
«Сравнение различных моделей машинного обучения и нейросетевых архитектур для решения задачи определения тональности текста»

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Машинное обучение (Machine Learning, ML) и искусственный интеллект (ИИ) сегодня стали неотъемлемой частью человеческой повседневности. Людям на выбор предлагается огромный спектр всевозможных услуг, технологий и приложений, использующих концепции ИИ; начиная от сервиса по автоматическому созданию презентаций, заканчивая полномасштабными языковыми моделями, решающими всевозможные задачи, и умными роботами-пылесосами.

В связи с такой нарастающей популярностью интеграции машинного обучения в различные аспекты жизни человека, также возрастает и потребность в накоплении всевозможных знаний и опыта в сфере применения тех или иных решений на базе концепции ML. Именно этому и посвящена данная работа. Фокус работы направлен в конкретном направлении ИИ — Native Language Preprocessing (NLP). Если говорить кратко и просто, данное направление изучает проблематику, способы обработки и генерации естественного языка компьютером.

В данной работе будет приведено сравнение различных нейросетевых архитектур и классических методов машинного обучения для решения задачи определения тональности текста.

Целью исследования является провести сравнительный анализ эффективности классических методов машинного обучения и нейросетевых архитектур по следующим критериям:

1. Качество модели — анализ метрик, насколько модель хорошо справляется с задачей определения тональности текста.
2. Скорость обучения.
3. Потребление ресурсов — CPU, GPU, память.
4. Зависимость от объема обучающей выборки — насколько модель чувствительна к количеству исходных данных.

5. Интерпретируемость модели — «черный ящик» или понятные зависимости.

Задачи данной работы:

1. Провести обзор существующих подходов для задачи определения тональности текста.
2. Проанализировать структуру и свойства выбранного датасета.
3. Реализовать и обучить модели каждого подхода на одном и том же датасете.
4. Оценить эффективность моделей по ключевым метрикам.
5. Сравнить результаты и сформулировать выводы о сильных и слабых сторонах каждого подхода.

Среди методов исследования используется следующие: анализ, синтез, экспериментальный метод, математико-статический анализ, сравнительный анализ.

1 АНАЛИТИЧЕСКИЙ ОБЗОР

1.1 Обзор подходов и решений

Так как работа проводится в области NLP, то объектом предобработки и анализа будет являться текст на естественном языке. Язык очень многогранная и сложная система, включающая себя грамматику (падежи, времена, склонения), пунктуацию, диалектизмы и многое другое. Таким образом, в данной области ИИ имеется огромное количество различных библиотек и подходов для обработки и анализа данных, и именно языка.

Задачи предобработки, библиотеки и методы их решения можно классифицировать следующим образом:

1. Очистка текста от мусорных слов и символов.

Включает в себя:

- удаление HTML-тегов;
- удаление пунктуации и специальных символов;
- удаление чисел, эмодзи, email-адресов, ссылок;
- удаление повторов и лишних пробелов;

Библиотеки и методы для решения данных задач:

- библиотека **re** для создания регулярных выражений;
- **beautifulSoup** (удаление HTML-тегов);
- **emoji**, **demojize** — работа с эмодзи.

2. Токенизация (Tokenization);

Что делает:

- процесс разделения текста на отдельные смысловые единицы (токены). Токены могут быть отдельными словами, или же набором слов (n-граммы).

Библиотеки:

- **nltk.word_tokenize**, **nltk.sent_tokenize;**
- **spaCy;**
- **razdel.**

3. Удаление стоп-слов.

Что делает:

- убирает распространенные слова, например «и», «в», «на», "the", "and".

Библиотеки:

- **nltk.corpus.stopwords.words('russian');**
- **spaCy;**
- **sklearn.feature_extraction.text;**
- **stopwords-iso** (поддержка многих языков).

4. Стемминг;

- обрезает слово до его основы или корня.

Библиотеки:

- **nltk.PorterStemmer()**;
- **nltk.PorterStemmer()** (для русского языка).

5. Лемматизация.

Принцип работы:

- чаще всего представляет собой процесс приведения слова к нормальной (неопределенной) форме. Например, «уронили» к «ронять», «бегут» к «бегать» и так далее.

Библиотеки:

- **nltk.WordNetLemmatizer()** (только английский);
- **spaCy**;
- **pymorphy2** (для русского языка);
- **Stanza** (универсальный парсер, включая лемматизацию).

6. Нормализация.

Включает в себя:

- приведение к нижнему регистру;
- замена сокращений;
- исправление орфографии.

Библиотеки и методы:

- **str.lower()** (если текст находится в столбце датафрейма из Pandas или представляет собой строку;
- **SymSpell, pyspellchecker, jamspell** — библиотеки для исправления ошибок;
- **textblob.correct()** (для английского языка).

В данном разделе был проведен обзор основных подходов и решений для задач, встречающихся в рамках предобработки текста на естественном языке.

1.2 Характеристика данных

1.2.1 Описание данных

В рамках исследовательской работы взят датасет с онлайн-ресурса и одноименной компании "Hugging Face", занимающейся разработкой и продвижением инструментов машинного обучения, в особенности в области обработки естественного языка (NLP). Датасет доступен по ссылке https://huggingface.co/datasets/MonoHime/ru_sentiment_datasethttps://huggingface.co/datasets/MonoHime/ru_sentiment_dataset.

Набор данных представляет собой совокупность из 210989 строк и является сборником комментариев пользователей различных ресурсов на разные темы. Какие это именно темы станет понятно далее. Стоит отметить, что датасет представляет собой сборник из шести отдельных аналогичных по структуре независимых наборов данных. Далее приведена таблица, отображающая первые пять и последние 5 строк данных (Таблица 1.1).

Таблица 1.1 — Демонстрация набора данных

Индекс	**Unnamed: 0**	**text**	**sentiment**
0	43956	Развода на деньги нет\nНаблюдаюсь в Лайфклиник...	1
1	17755	Отель выбрали потому что рядом со стадионом. О...	0
2	20269	Вылечили\nГноился с рождения глазик, в поликли...	1
3	16648	Хорошее расположение.С вокзала дошли пешком.Но...	0
4	27879	Отличное месторасположение,прекрасный вид,особ...	1
…	…	…	…
210984	22100	Мой юбилей я отмечал в ресторане " Астория "	2
210985	2326	Отлично встретили, разместили в роскошном номе...	1
210986	10478	Была в Васаби на ст. метро Сенная . Во первых...	0
210987	4028	Ребята не стоит смотреть этот фильм. Вы молоды...	0

Набор данных состоит из следующих колонок (признаков):

1. Индекс — Автоматически генерируемая структура, по умолчанию входящая в любой набор данных, представленный в виде датафрейма (DataFrame) библиотеки Pandas.
2. "Unnamed: 0" — Поле, не совсем ясного назначения, вероятно оно обозначало идентификатор пользователя, оставлявшего комментарий. Его назначение станет ясно позже.
3. "text" — Непосредственно сам текст и комментарии, поле, представляющее для исследования наибольшую ценность и являющееся основным признаком для дальнейшей работы.
4. "sentiment" — Оценка тональности текста, где:

- 0: нейтральная оценка;
- 1: позитивная оценка;
- 2: негативная оценка;

Ниже приведен Листинг 1.1 с описанием датасета в виде структуры данных DataFrame из библиотеки Pandas.

Листинг 1.1 — Общее описание набора данных

```
# Импортируем все необходимые библиотеки для этого
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

df_datasets = pd.read_csv('datasets.csv') # Полный набор данных


-> <class 'pandas.core.frame.DataFrame'>
-> RangeIndex: 210989 entries, 0 to 210988
-> Data columns (total 3 columns):
-> #   Column      Non-Null Count   Dtype
-> ---  ------      --------------   -----
-> 0   Unnamed: 0  210989 non-null  int64
-> 1   text        210989 non-null  object
-> 2   sentiment   210989 non-null  int64
-> dtypes: int64(2), object(1)
-> memory usage: 4.8+ MB
```

Здесь и далее символом "->" помечается результат выполнения функций и программ. Как можно увидеть из отчета о наборе данных, пропущенных значений не обнаружено. Имея достаточно наглядное описание данных, можно приступить к детальному анализу датасета.

1.2.2 Анализ набора полей данных

В данном подразделе будет установлено назначение неизвестных полей и проанализированы известные. Надо сказать, что полей в наборе данных не так много.

Выявления наличия уникальных значений у поля "Unnamed: 0" поможет выяснить является ли это поле каким-то уникальным идентификатором, несущим какую-либо полезную информацию (Листинг 1.2).

Листинг 1.2 — Проверка на уникальность поля "Unnamed: 0"

```
len(df_datasets) # Длина всего датафрейма
-> 210989

len(df_datasets['Unnamed: 0'].unique())
-> 70597

# не является уникальным идентификатором
```

Продолжение Листинга 1.2

```
df_datasets = df_datasets.rename(columns={'Unnamed: 0' : 'Col1'})
# Переименовываем для удобства

# Другой способ

df_datasets.Col1.value_counts()

-> Col1
-> 3853     8
-> 30       8
-> 4930     8
-> 1742     8
-> 4503     8
->          ..
-> 54039    1
-> 67553    1
-> 67236    1
-> 53702    1
-> 57358    1
-> Name: count, Length: 70597, dtype: int64
```

Как видно из листинга, поле "Unnamed: 0" не является уникальным, и многие его элементы повторяются. Данное поле не несет никакого практического смысла и его можно удалить.

Поле "text" представляет собой сырой текст, который может содержать различные мусорные слова, такие как:

- управляющие последовательности: специальные технические символы, регулирующие отображение текста, отвечающие за табуляцию, перемещение каретки, новый абзац и так далее;
- ссылки на веб-ресурсы;
- номера телефонов;
- почтовые адреса;
- различного рода смайлы и эмодзи;
- видоизмененные слова или редкие слова.

Так как просмотреть содержимое всех текстов не представляется возможным, то нужно учесть наиболее широкий спектр всевозможных мусорных слов, не несущих никакого смысла и только усложняющих и ухудшающих предсказательную способность модели. Этот процесс называется предобработкой текста, и он является основополагающим в задачах NLP.

Далее будет представлен Листинг 1.3, содержащий весь необходимый код для преобразования данных поля "text".

Листинг 1.3 — Процесс предобработки текста

```
# Импортируем все необходимые библиотеки

import re # Библиотека для написания регулярных выражений
import nltk # Загружаем библиотеку для обработки языка Natural Language Toolkit
from nltk.corpus import stopwords
import pymorphy3 # Библиотека для лемматизации

morph = pymorphy3.MorphAnalyzer() # В данном случае слова будут приводиться к нормальной (неопределенной) форме
stop_words = set(stopwords.words('russian')) # Инициализируем словарь стоп-слов для русского языка

# Чуточку подкорректируем его
stop_words.remove('хорошо')

      # Лучше сделать все в одной большой функции

      def text_preprocessing(df: pd.DataFrame, colname: str) -> pd.DataFrame:

          """Выполняет полную предобработку текста:
          нормализацию, очистку, лемматизацию и тд."""

          df[colname] = (
              df[colname]

              # Преобразуем строки в нижний регистр
              .astype(str).str.lower()

              # Удаление смайликов
              .str.replace(r"[():]", " ", regex=True)

              # Удаление различных знаков припинания
              .str.replace(r"[^\w\s]", "", regex=True)

              # Удаление управляющих символов в кодировке ASCII
              .str.replace(r'[\x00-\x1F\x7F]', ' ', regex=True)

              # Удаление mail-адресов
              .str.replace(r"[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}",
                           "",
                           regex=True)

              # Удаление номеров телефонов
              .str.replace(r"(?:\+7|8)[-.,']?\(?\d{3}\)?[-.,']?\d{3}"
                           r"[-.,']?\d{2}[-.,']?\d{2}",
                           "",
                           regex=True)

              # Удаление ссылок
              .str.replace(r"https?://[^\s]+|www\.[^\s]+|\b[a-zA-Z0-9.-]+\."

r"(com|ru|net|org|info|biz|io|dev|gov|edu|co|uk|ua|de|fr)\b",
                           "",
                           regex=True)
```

Продолжение Листинга 1.3

```
            # Удаление дат
            .str.replace(r"\b(?:\d{1,2}[-/.\s])?(?:\d{1,2}[-
/.\s])?\d{2,4}\b",
                         "",
                         regex=True)

            # Удаление стоп-слов
            .apply(lambda x:
                   '  '.join([str(i)  for  i  in  x.split()  if  i  not  in
stop_words]))

            # Лемматизация
            .apply(lambda x:
                   '  '.join([morph.parse(word)[0].normal_form  for  word  in
x.split()]))
           )
        return df
```

Этот код предназначен для полной предобработки текстов на русском языке перед их использованием в моделях обработки естественного языка, таких как определение тональности текста. Сначала осуществляется импорт необходимых библиотек: ранее упомянутая **re** для работы с регулярными выражениями, **nltk** — для загрузки списка стоп-слов на русском языке, а **pymorphy3**, как уже было оговорено, применяется для лемматизации, то есть приведения слов к их начальной форме. Инициализируется морфологический анализатор **pymorphy3.MorphAnalyzer()** и загружается набор стоп-слов из библиотеки **nltk**, при этом вручную исключается слово «хорошо», так как оно может нести важную смысловую нагрузку при анализе тональности.

Основная часть логики реализована в виде функции **text_preprocessing**, которая принимает на вход датафрейм и имя колонки, содержащей тексты. Эта функция обрабатывает тексты поэтапно: сначала преобразует все символы к нижнему регистру, затем удаляет смайлики, знаки препинания, управляющие символы ASCII, электронные адреса, номера телефонов, гиперссылки и даты. После этого из текста удаляются все стоп-слова, за исключением ранее сохранённого слова «хорошо». На последнем этапе происходит лемматизация текста: каждое слово преобразуется к его нормальной (начальной) форме. Вся обработка осуществляется с помощью встроенных инструментов библиотеки Pandas, что позволяет выполнять преобразования эффективно и векторизованно,

так как функция **str.replace** оптимизирована на уровне C. Результатом работы функции является датафрейм с очищенным, нормализованным и лемматизированным текстом, который может быть использован для дальнейшего анализа или подачи на вход модели машинного обучения.

Надо сказать, незначительная часть строк в наборе данных стала содержать значение None (12 строк), так что это значение настолько незначительно, что им спокойно можно пренебречь и избавиться от данных «испортившихся» строк.

На Рисунке 1.1 для наглядности работы функции, показано сравнение одной преобразованной строки текста и одной строки сырого текста.

```
Ввод [250]: df_datasets_prep.text.loc[1010]

Out[250]: 'ортопед поменять прокошина пенсия пора родитель унижать говорить повышенный тон пугать сколиоз так д
          алее спорить родитель любой повод пена рот господь поликлиника быть вменяемый врач'

Ввод [252]: df_datasets.text.loc[1010]

Out[252]: 'Ортопеда поменяйте\nПрокошиной на пенсию пора! Родителей унижает, говорит только на повышенных тона
          х, всех пугает сколиозом и т. д. Спорит с родителями по любому поводу с пеной у рта. Господи, когда в
          этой поликлинике будут вменяемые врачи?\n'
```

Рисунок 1.1 — Демонстрация преобразования текста

Третья колонка "sentiment", являющаяся в данной задаче целевой переменной, как уже было изложено выше, состоит из трех значений, определяющих тональность конкретного текста. Визуализация соотношения классов в наборе данных поможет лучше понять суть данных и станет важным дополнением для выбора конкретного алгоритма машинного обучения с целью

решения задачи выявления тональности текста (Рисунок 1.2).

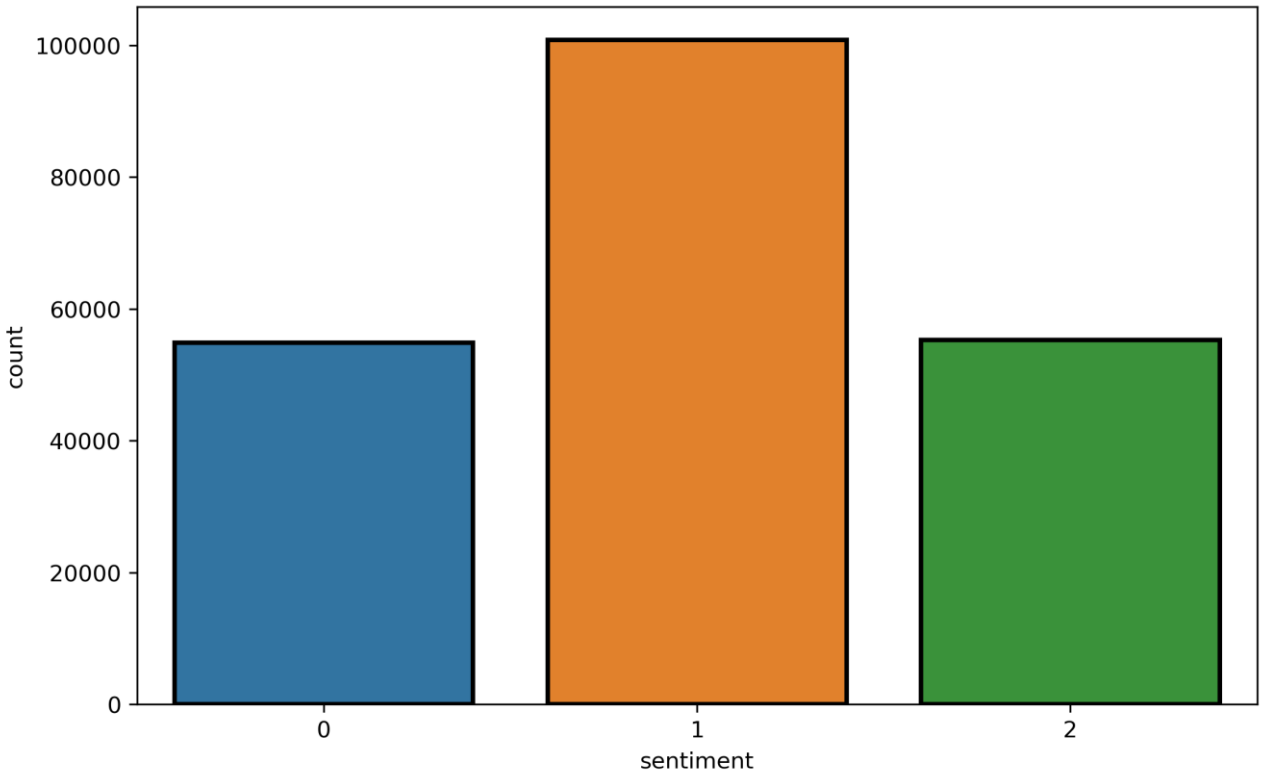

Рисунок 1.2 — Иллюстрация соотношения классов тональности в наборе данных

Как видно из графика, 1 класс преобладает над остальными, то есть классы слегка несбалансированы, что может впоследствии вызвать некоторые проблемы при обучении, такие как переобученность, если не предпринять никаких мер.

1.2.3 Тематическое моделирование текстовых данных

Тематическое моделирование (topic modeling) в задачах NLP — это метод автоматического выявления скрытых тем в коллекции текстов. Оно нужно для того, чтобы понять, **о чём идет речь** в большом объёме текстовых данных без ручного чтения, а также для **кластеризации**, **сжатия информации**, **поиска схожих документов** и **визуализации структуры данных**.

Чаще всего тематическое моделирование используется для:

- анализа новостных архивов, отзывов, документов;
- выявления ключевых направлений обсуждений;

- предварительного анализа перед обучением других моделей.

Существует множество подходов и библиотек на различных языках программирования для выполнения данной задачи. Так как работа проводится в рамках Python, то будут рассматриваться решения в пределах данного языка. Наиболее популярные из них:

1. LDA (Latent Dirichlet Allocation) — вероятностная генеративная модель, которая представляет документы как смеси тем, а темы — как распределения слов. Использует метод Дирихле для определения скрытых (латентных) тем в тексте.

Плюсы:

- хорошо работает с длинными текстами (например, статьи, отзывы);
- прозрачная интерпретация тем через ключевые слова;
- имеет множество реализаций в различных библиотеках (реализован в **genism**, **scikit-learn**).

Минусы:

- требует предварительной обработки текста (удаление стоп-слов, лемматизация);
- чувствителен к гиперпараметрам;
- плохо работает с короткими текстами.

2. NMF (Non-negative Matrix Factorization) — Линейно-алгебраический метод, разлагающий матрицу «документы-термины» на две неотрицательные матрицы: «документы-темы» и «темы-термины».

Плюсы:

- лучшая интерпретируемость тем по сравнению с LDA (более четкие ключевые слова);
- эффективен для коротких текстов при использовании TF-IDF (способ векторизации текста);
- меньше зависимости от гиперпараметров, чем в LDA.

Минусы:

- требует ручного выбора числа тем (нет встроенной оптимизации);
- менее точен на очень длинных текстах.

3. BERTopic — Использует BERT (одна из самых распространенных архитектур нейросетей, называемая трансформером) для векторного представления текста и кластеризацию (например, UMAP + HDBSCAN) для выделения тем. Темы описываются через ключевые слова из кластеров.

Плюсы:

- автоматическое определение числа тем;
- отлично работает с короткими текстами;
- минимальная предобработка;
- поддержка иерархического тематического моделирования.

Минусы:

- ресурсоёмкий (требует GPU для больших данных);
- сложность интерпретации для узкоспециализированных текстов.

Таким образом, наилучшим методом тематического анализа в данной ситуации среди представленных можно назвать **LDA** по следующим причинам:

1. Часто комментарии в сети представляют собой смесь разнообразных тем и предметных областей. LDA хорошо работает с такими случаями, поскольку сам алгоритм предполагает, что документ представляет собой совокупность тем и оценивает вероятность принадлежности конкретного документа к той или иной теме.
2. LDA хорошо работает с большим корпусом данных. Представленный корпус данных довольно крупный, как и входящие в него документы (некоторые комментарии могут достигать вдлину 200-300 слов). Такого объема вполне достаточно для эффективной работы данного алгоритма тематического моделирования.

3. За счет вероятностной природы, данный алгоритм также является высокоинтерпретируемым и имеет множество удобный библиотек и инструментов для его визуализации.

BERTopic не подходит по следующим причинам:

1. Требовательность к ресурсам — BERTopic основан на особой архитектуре нейронных сетей под названием трансформеры, это очень требовательные в вычислительном смысле нейронные сети, использующие представление слова в виде длинного вектора, еще называемого эмбеддингом (emdedding). Создание такого эмбеддинга очень затратная операция, а слов тысячи.
2. Очень сложное и избыточное решение для реализуемой в данной работе задачи.

NMF можно было бы использовать, если бы тексты были короче (на них данный метод работает наиболее точно) и если бы была использована реализация BoW (Bag of Words) с использованием TF-IDF (Term Frequency – Inverse Document Frequency) — особый вид векторизации, заключающийся в умножении частоты встречаемости слова в определенном документе на коэффициент, равный отношению количества документов, в которых присутствует данное слово, ко всем документам данного корпуса.

Ниже представлен Листинг 1.4, содержащий код для создания мешка слов (Bag of Words), который затем подается на алгоритм LDA для высчитывания вероятности принадлежности документа к той или иной теме и принадлежности темы к определенным словам соответственно, после чего приводится код для создания визуализации результата работы данного алгоритма.

Листинг 1.4 — Процесс предобработки текста

```
from gensim import corpora

# Создаем словарь всех слов в корпусе
dictionary = corpora.Dictionary(texts)

# Удаление редких и неиспользуемых слов
dictionary.filter_extremes(no_below=10, no_above=0.5)

# Создаем Bag of Words
```

Продолжение Листинга 1.4

```
BoW = [dictionary.doc2bow(text) for text in texts]

from gensim.models.ldamulticore import LdaMulticore

# Задаем параметры LDA
lda_model = LdaMulticore(
    corpus=BoW,
    id2word=dictionary,
    num_topics=10,     # Кол-во тем (можно подбирать вручную)
    passes=10,         # Кол-во проходов по корпусу
    iterations=100,
    chunksize=2200,    # Размер обрабатываемой порции данных за раз (чанк)
    workers=2,         # Кол-во используемых логических ядер процесса
    #alpha='auto',     # Параметр разреженности распределения слов (выкл.)
    #eta='auto',
    random_state=42,
    minimum_probability=0.01,
    per_word_topics=True # Сохранять информацию о вероятностях слов в темах
) # Обучение непосредственно алгоритма


# Визуализация

import pyLDAvis.gensim_models as gensimvis
import pyLDAvis

# Подготовка визуализации
vis_data = gensimvis.prepare(lda_model, BoW, dictionary)

# Отображение в Jupyter или сохранение в HTML
pyLDAvis.display(vis_data)


pyLDAvis.save_html(vis_data, 'lda_visualization.html')
```

Далее на Рисунках 1.3 — 1.12 приведены визуальные представления тем и самые релевантные слова, встречающиеся в этих темах.

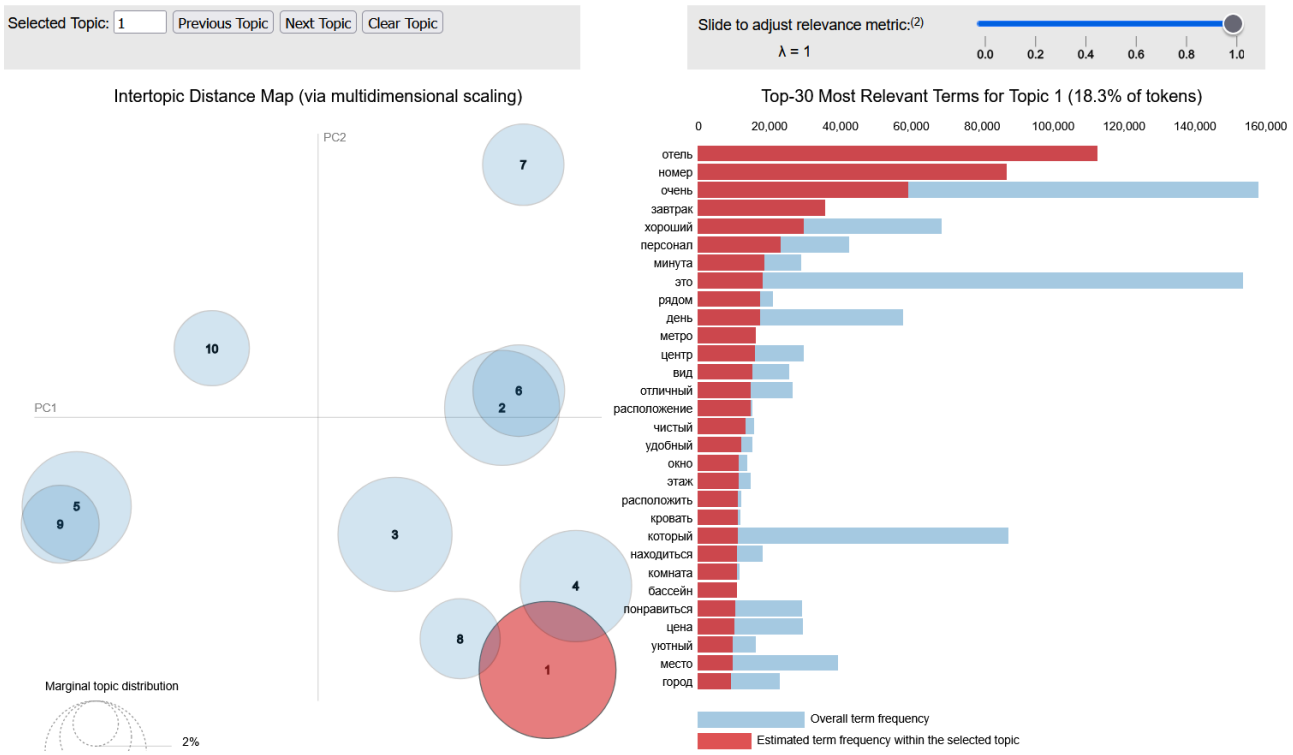

Рисунок 1.3 — Визуализация Тема 1

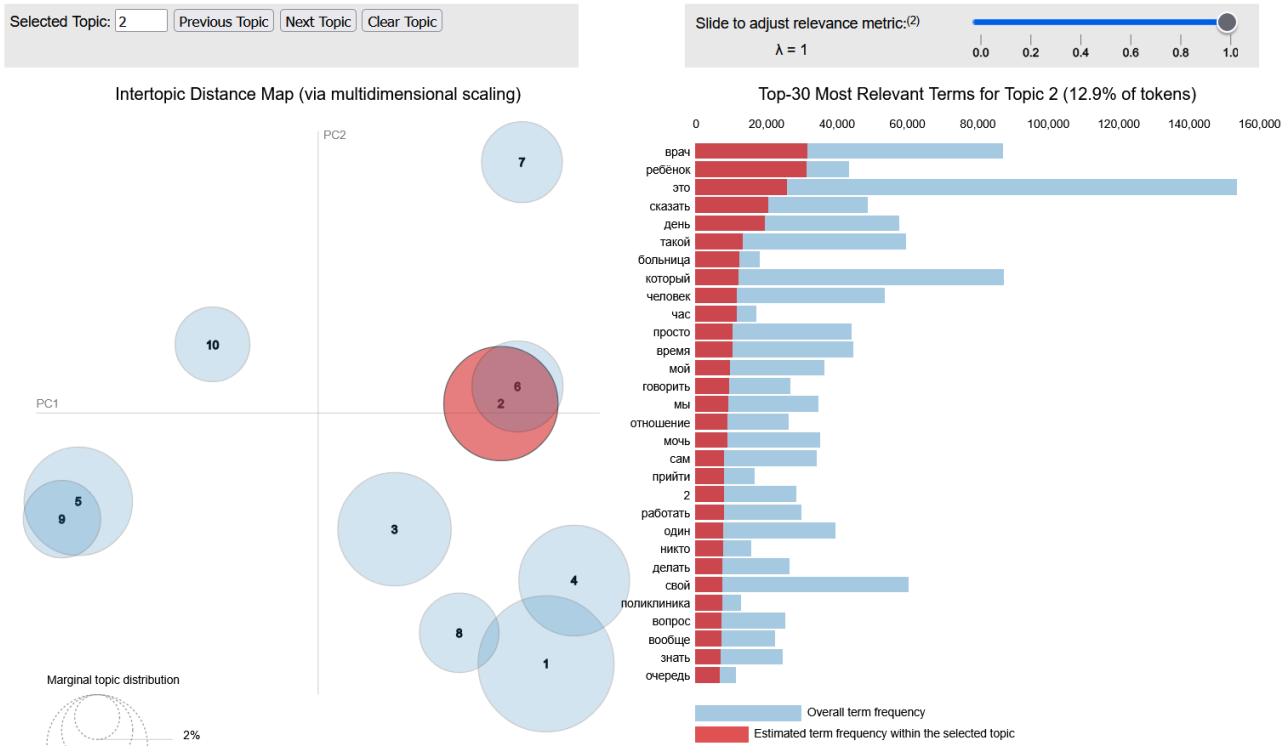

Рисунок 1.4 — Визуализация Тема 2

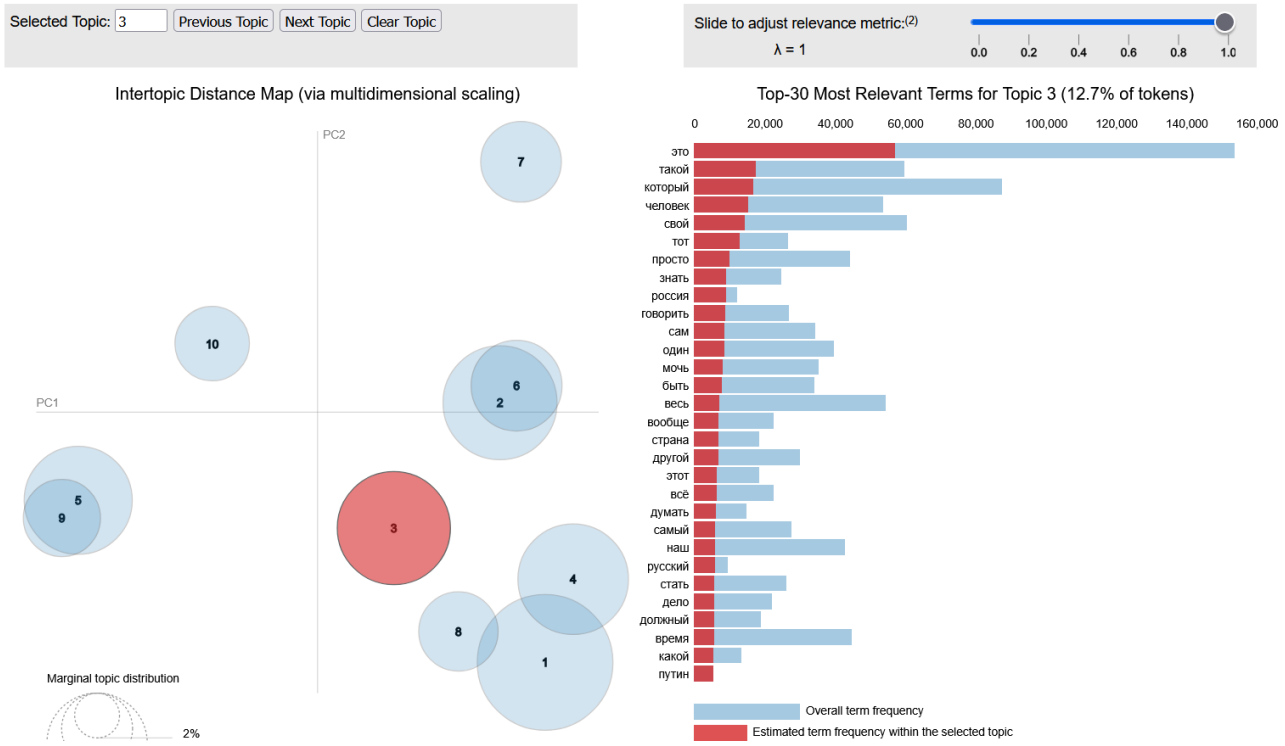

Рисунок 1.5 — Визуализация Тема 3

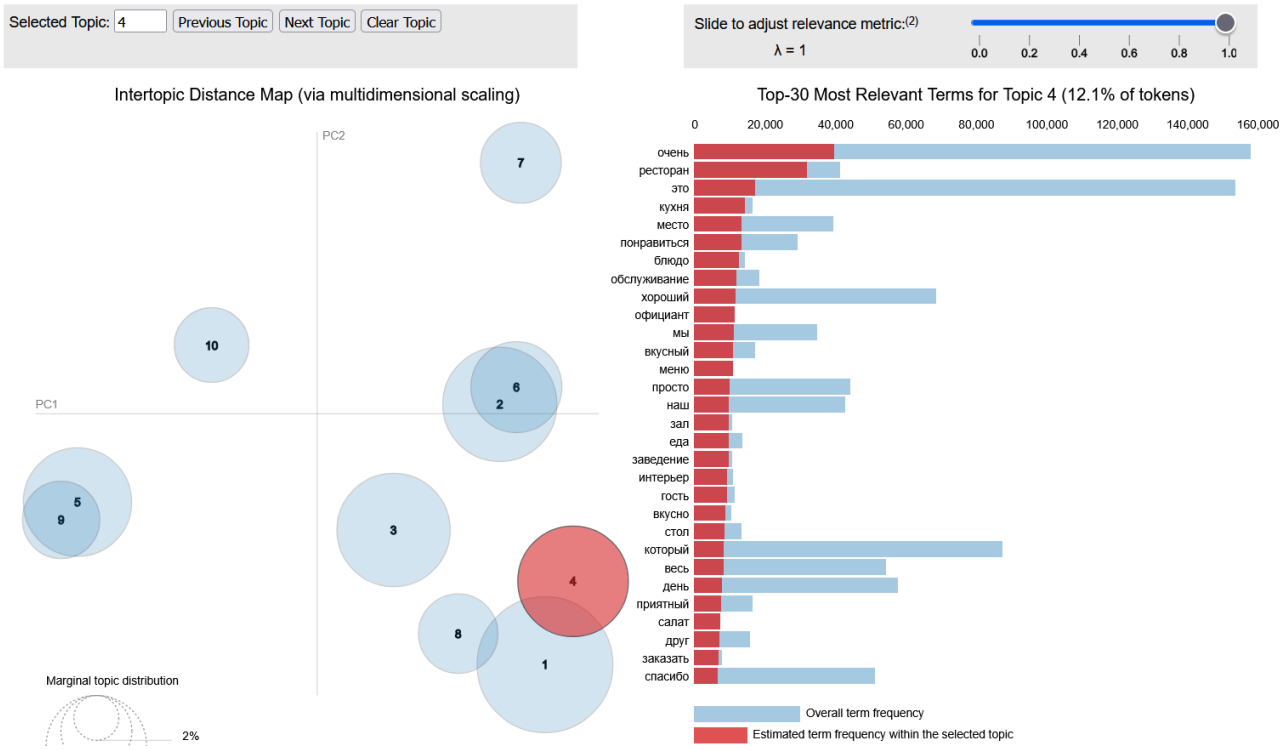

Рисунок 1.6 — Визуализация Тема 4

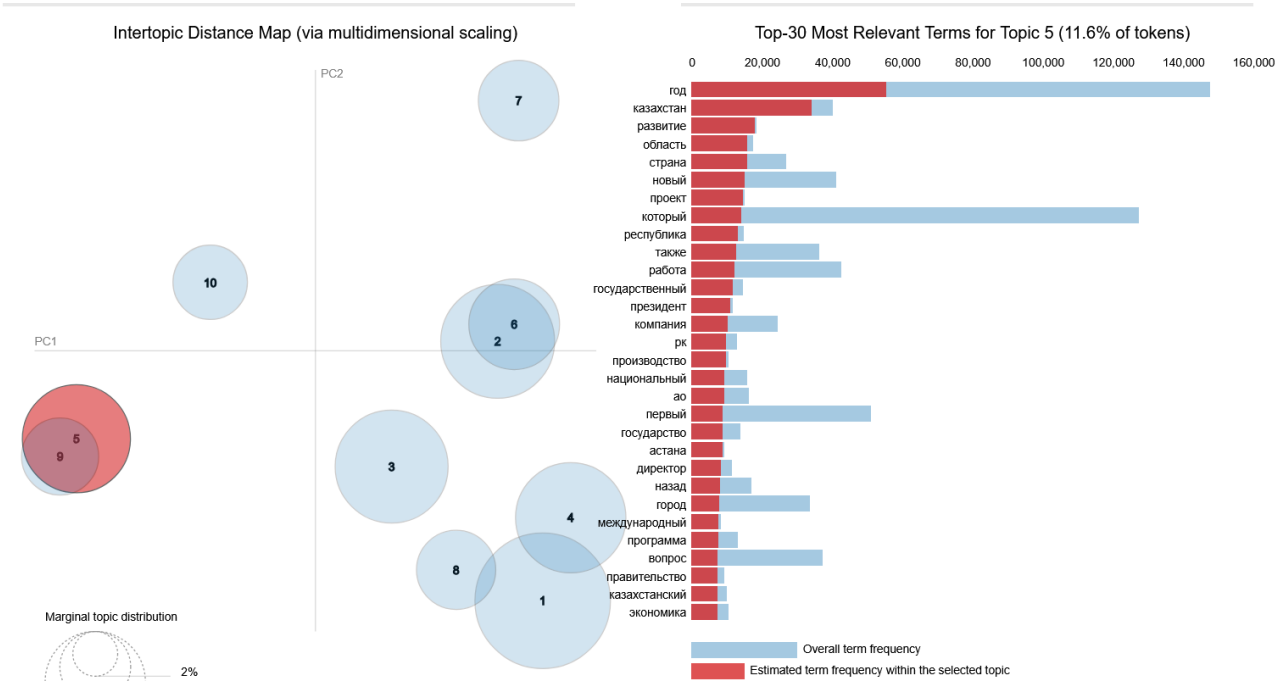

Рисунок 1.7 — Визуализация Тема 5

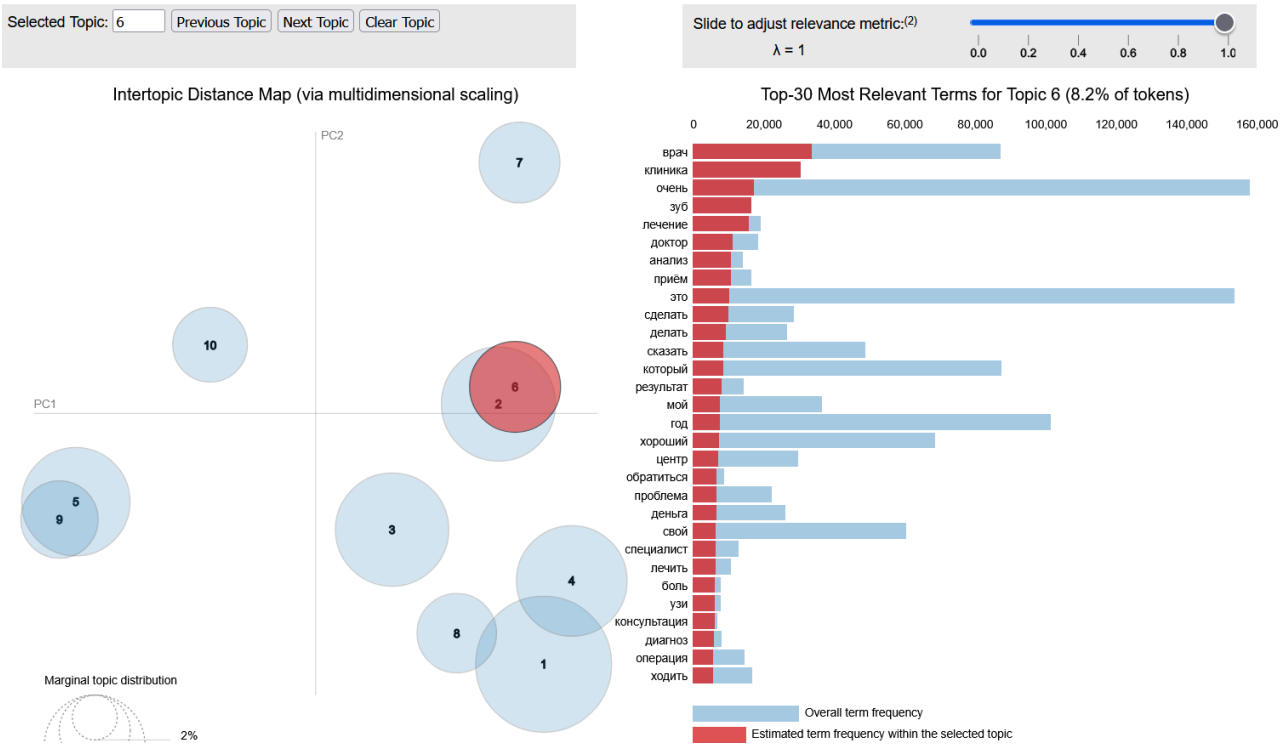

Рисунок 1.8 — Визуализация Тема 6

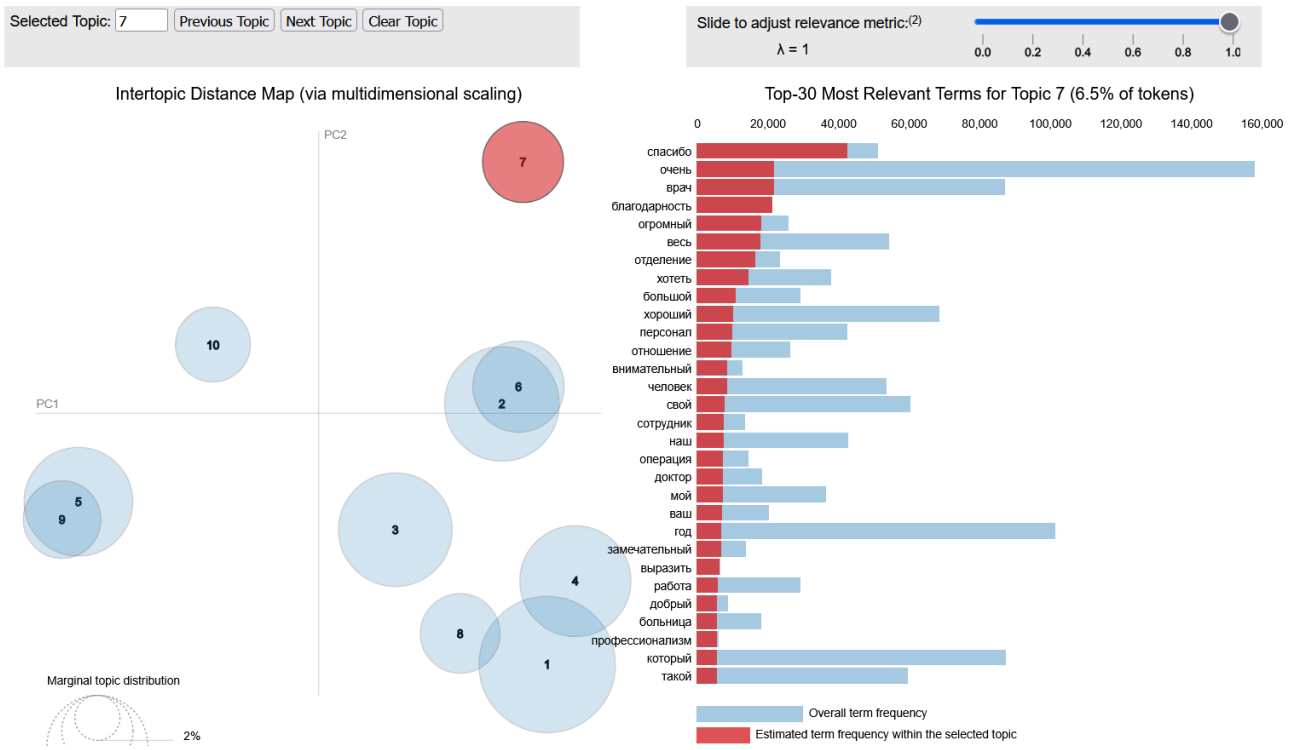

Рисунок 1.9 — Визуализация Тема 7

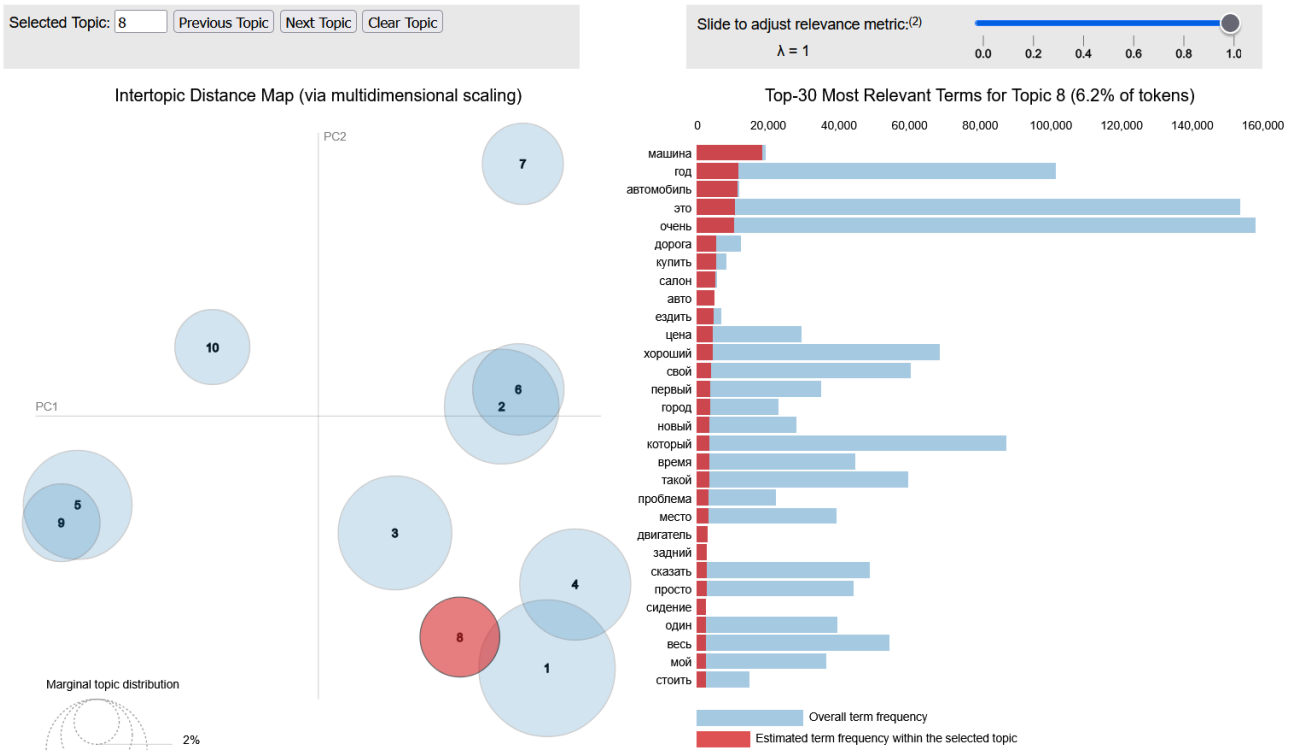

Рисунок 1.10 — Визуализация Тема 8

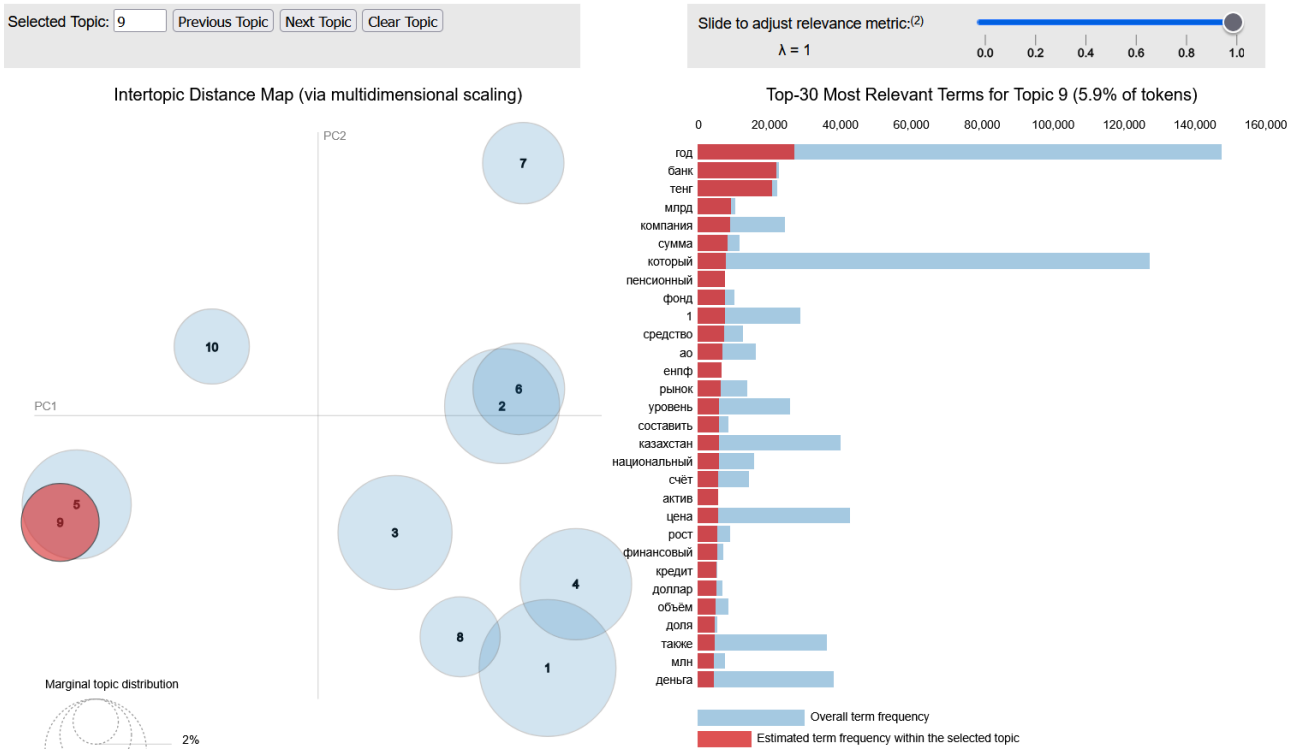

Рисунок 1.11 — Визуализация Тема 9

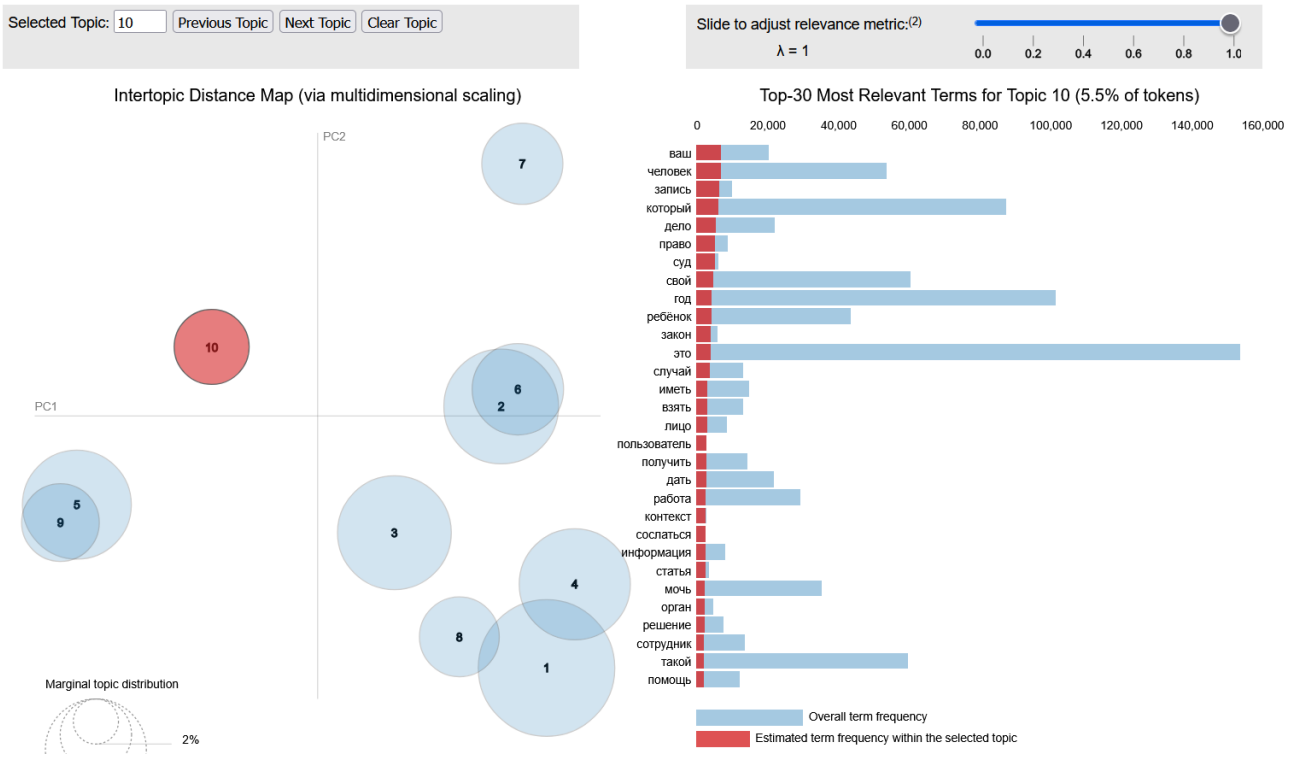

Рисунок 1.12 — Визуализация Тема 10

Исходя из следующих изображений визуализаций можно заключить, что в основном темы сконцентрированы вокруг нескольких главных предметных областей — гостиничное дело (Тема 1), ресторанное дело (Тема 4), темы на

автомобильную тематику (Тема 8), политика (Тема 3), здравоохранение (Темы 2, 6, 7), финансы и юриспруденция (Темы 9, 10), а также Казахстан (Тема 5). Надо сказать, что темы 2, 6 и 7 можно объединить в одну, путем уменьшения числа тем, однако и такое количество тем по умолчанию позволяет оценить содержание, направленность и настроение тем в данном корпусе слов.
Следующим шагом будет непосредственное обучение и сравнение определенных моделей машинного обучения и нейросетевых архитектур.

2.3 Постановка задачи

В рамках всей работы, на основе полученных знаний и предобработанных данных, предполагается обучить ряд классификаторов для определения тональности текста, основанных на классических алгоритмах машинного обучения и нейросетевых архитектурах.

Это позволит сравнить различные подходы и технологии по ряду критериев таких, как: точность предсказаний, скорость выполнения, скорость обучения (на GPU, CPU) и оптимизация.

2 ПРОЕКТНАЯ ЧАСТЬ

2.1 Модели классического машинного обучения

В данном подразделе будут рассмотрены модели классического машинного обучения, такие как логистическая регрессия и различные ансамбли на основе решающих деревьев. Особенный акцент будет направлен на разные реализации градиентного бустинга, так как общая практика и опыт их применения в моделях классификации, в том числе и в определении тональности текста, продемонстрировал высокую эффективность данного вида ансамблей, по сравнению с другими ансамблевыми методами, такими как, например, бэггинги.

В качестве метода представления набора документов в корпусе в цифровом виде, главным образом, будет использована концепция BoW (Bag of Words) в виде реализации TF-IDF. Такое решение обусловлено следующим причинами:

1. Учет важности слов:

- TF-IDF уменьшает вес часто встречающихся слов (например, стоп-слов) и увеличивает значимость редких, но информативных терминов, которые могут влиять на тональность (например, "ужасный", "превосходный").

2. Простота и интерпретируемость:

- Метод легко реализуется и интерпретируется, в отличие от сложных нейросетевых подходов. Можно анализировать, какие слова имеют наибольший вес в классификации.

3. Работает с небольшими датасетами:

- В отличие от методов, требующих больших объемов данных (например, word2vec, BERT), TF-IDF может давать хорошие результаты даже на малых корпусах.

4. Нормализация частоты слов:

- Уменьшает влияние длинных документов за счет нормировки на количество слов (TF), что полезно, если тексты в корпусе разной длины.

5. Совместимость с классическими ML-моделями:

- После векторизации можно использовать линейные модели (логистическая регрессия, бустинги, решающие деревья, SVM), которые часто хорошо работают в задачах классификации текстов.

Конечно данный подход имеет и недостатки, такие как:

1. Игнорирование контекста и семантики — Слова представлены в виде неупорядоченного набора векторов, где строки это конкретные документы, а столбцы вся совокупность слов, встречаемых в данном корпусе документов и значения в данной матрице лишь обозначают отсутствие или присутствие конкретного слова с коэффициентом, обратным частоте появления данного слова в документах корпуса (Inverse Document Frequency). То есть данные слова никак не связаны между собой и не учитывают ни семантики ни контекста.
2. Размеры представления — BoW в виде TF-IDF является большой разреженной матрицей и количество признаков в такой матрице равняется размеру словаря данного корпуса слов. Другими словами признаками могут являться и отдельные уникальные слова, входящие в совокупность всех слов, и их отдельные словосочетания (n-граммы). Обучение на таких больших структурах может быть ресурсоемким, но все равно не таким ресурсоемким как использование нейросетевых подходов.

Таким образом, TF-IDF был выбран за свою простоту и интерпретируемость, по сравнению с нейросетевыми подходами, а также в связи с тем, что является наиболее удачным представлением данных для подачи их на вход моделям классического машинного обучения.

2.1.1 Логистическая регрессия

Логистическая регрессия — это статистический метод классификации машинного обучения. Сравнительно простая модель.

Несмотря на название, логистическая регрессия не является регрессионной моделью в классическом смысле, а представляет собой метод бинарной классификации. Модель предсказывает вероятность принадлежности наблюдения к одному из двух (или множества) классов, используя логистическую функцию активации, которая отображает любое действительное значение в диапазон [0,1].

Пусть $\mathrm{x} = [x_1, x_2, \dots, x_n]$ — вектор признаков. Задача логистической регрессии — аппроксимировать вероятность того, что клиент относится к тому или иному классу, через следующую функцию:

$$P(y = 1|x) = \frac{1}{1 + e^{-z}}, \tag{2.1}$$

Где

$$z = \beta_0 + \beta_1 \cdot x_1 + \cdots + \beta_i \cdot x_i, \tag{2.2}$$

Здесь:

- β_0 — свободный член (intercept);
- β_i — весовые коэффициенты при признаках;
- e — основание натурального логарифма;
- $\mathrm{P}(\mathrm{y} = 1 \mid \mathrm{x})$ — вероятность положительного исхода.

Если взять отношение полученной вероятности к её комплементу и прологарифмировать, получаем логит:

$$ln(\frac{P}{1 - P}) = \beta_0 + \beta_1 \cdot x_1 + \cdots + \beta_n \cdot x_n, \tag{2.3}$$

Таким образом регрессия линейно моделирует именно логарифм шансов (odds) события.

Обучение логистической регрессии заключается в нахождении таких коэффициентов β_i, которые наилучшим образом соответствуют обучающей

выборке. В отличие от линейной регрессии, где используется метод наименьших квадратов, в логистической регрессии применяется максимизация функции правдоподобия (Maximum Likelihood Estimation, MLE).

Коэффициенты β выбирают методом максимального правдоподобия:

$$\mathrm{L}(\beta) = \sum_{i=1}^{m} [y_i \cdot \ln(p_i) + (1 - y_i) \cdot \ln(1 - p_i)], \quad (2.4)$$

Где

$$p_i = (1 + e^{-x_i \cdot \beta})^{-1}, \quad (2.5)$$

Задача — найти такое β, при котором функция $L(\beta)$ достигает максимума. Максимум L ищут численно: градиентным спуском, Newton-Raphson или L-BFGS; последний выбран «по умолчанию» в scikit-learn, потому что хорошо работает на умеренно-разреженных таблицах, типичных для TF-IDF матриц.

Ниже будет представлена таблица с характеристиками устройства, на котором будет производиться обучение моделей (Таблица 2.1).

Таблица 2.1 — Характеристики устройства, на котором производится обучеие

Комплектующее	**Характеристика**
ОЗУ	8192 МБ
Доступная физическая память:	6 094 МБ
Виртуальная память: максимальный размер	923 МБ
Виртуальная память: Доступна	20 132 МБ
Виртуальная память: Используется	7 991 МБ
Процессор	Число процессоров - 1. [01]: AMD64 Family 23 Model 24 Stepping 1 AuthenticAMD ~2600 МГц
Видеокарта	NVIDIA GeForce MX230
Файл подкачки	20 131 МБ

Далее на Листинге 2.1 продемонстрирован код подбора гиперпараметров и обучения модели логистической регрессии, с разбиением предобработанных данных на тренировочную и тестовую выборку, преобразованием текста в мешок слов TF-IDF и последующим обучением модели.

Листинг 2.1 — Подбор гиперпараметров и обучение модели логистической регрессии

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, classification_report,
ConfusionMatrixDisplay
from sklearn.model_selection import GridSearchCV
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

# Задание векторизатора

vectorizer = TfidfVectorizer(max_features=35000, ngram_range=(1,2))

# Векторизуем данные

X_train_tfidf = vectorizer.fit_transform(X_train)
X_test_tfidf = vectorizer.transform(X_test)

# Объявление модели
    log_model = LogisticRegression(
    class_weight='balanced',
    n_jobs=-1,
    C=5,
    solver='saga',
    penalty='l2',
    max_iter=1000
)

start_time = time.time()  # Засекаем начальное время

log_model.fit(X_train_tfidf, y_train)

end_time = time.time()  # Засекаем конечное время
execution_time = end_time - start_time

print(f"Код выполнился за {execution_time:.4f} секунд")

# Визуализация данных

      def model_evaluate(model, X_test, y_test, target_names=None,
png_name='Матрица ошибок'):
          """
          Полная оценка модели классификации с визуализацией.

          Параметры:
          model - обученная модель sklearn
          X_test - тестовые данные
          y_test - истинные метки теста
          target_names - список названий классов (опционально)
          """
          # Получаем предсказания
          y_pred = model.predict(X_test)

          # Генерируем отчёт о классификации
          report = classification_report(y_test, y_pred,
target_names=target_names)

          # Строим матрицу ошибок
          conf_matrix = confusion_matrix(y_test, y_pred)

          # Настраиваем визуализацию
          plt.figure(dpi=300, figsize=(8, 6))
          if target_names is None:
```

Продолжение Листинга 2.1

```
            target_names = [f"Class {i}" for i in range(len(model.classes_))]

        # Визуализация матрицы ошибок
        disp = ConfusionMatrixDisplay(confusion_matrix=conf_matrix,
                                      display_labels=target_names)
        disp.plot(cmap='binary', values_format='d', ax=plt.gca())
        plt.title("Матрица ошибок", pad=20, fontsize=16)
        plt.xticks(rotation=45 if len(target_names[0]) > 10 else 0)  # Поворот
подписей при необходимости
        plt.tight_layout()
        plt.savefig(fname=f"{png_name}.png")

        print("Отчёт о классификации:\n")
    print(report)

# Вызов функции для расчета метрик качества модели

model_evaluate(log_model, X_test_tfidf, y_test, ['Нейтральный', 'Позитивный',
                            'Негативный'], png_name='Матрица ошибок логрег'
```

Путем тщательного перебора соответствующих гиперпараметров модели, было принято решение остановиться на тех, которые представлены в Листинге 2.1. Обучение модели заняло 17.8240 секунд. Смена метода регуляризации на L1 не давала значительного прироста качества модели, зато увеличивала время обучения модели до 20-30 минут.

Не способствовало значительному улучшению качества также и увеличение максимального количества признаков в матрице TF-IDF, как и введение биграмм и триграмм.

На Рисунках 2.1-2.2 продемонстрированы отчет о классификации с соответствующими метриками классификации и матрица ошибок, дающая наглядное представление о качестве модели.

Отчёт о классификации:

	precision	recall	f1-score	support
Нейтральный	0.59	0.73	0.65	5488
Позитивный	0.84	0.72	0.78	10079
Негативный	0.78	0.80	0.79	5531
accuracy			0.74	21098
macro avg	0.74	0.75	0.74	21098
weighted avg	0.76	0.74	0.75	21098

Рисунок 2.1 — Отчет о классификации для логистической регрессии

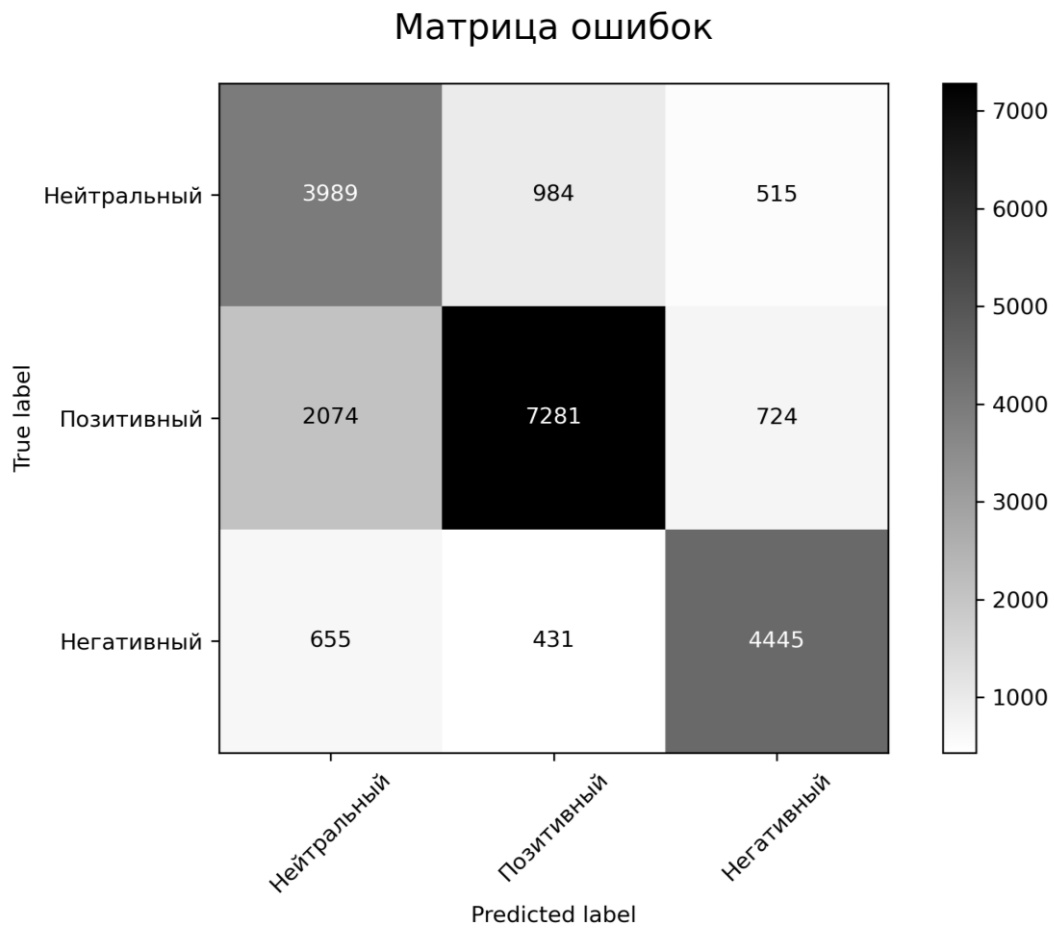

Рисунок 2.2 — Матрица ошибок для логистической регрессии

В целом модель демонстрирует неплохие показатели в определении Негативных и Позитивных текстов, однако качество падает на определении Нейтральных классов. Это можно увидеть по низкому **precision** для нейтрального класса, то есть модель Нейтральный класс часто принимает за какой-то другой из двух классов, а конкретно за Положительный. Это можно уже увидеть как и по матрице ошибок (значение 2074 на перекрестке Позитивного класса на оси True label и Нейтрального на оси Predicted label), как и по значению **recall** Положительного класса.

Это достаточно ожидаемый и понятный результат — модель всегда будет определять хуже именно Нейтральный класс, потому что он на то и нейтральный, что в нем нет резких и очевидных слов, которые могли бы его характеризовать, как, например, в Положительном («хорошо», «отлично») и в Негативном («плохо», «ужасно»).

2.1.2 Бустинг (LightGBM)

LightGBM — это фреймворк для градиентного бустинга на деревьях от Microsoft, оптимизированный для высокой скорости и эффективности. Он использует ансамблевый метод, где каждое новое дерево корректирует ошибки предыдущих, но с ключевыми улучшениями по сравнению с классическим GBDT (Gradient Boosting Decision Trees).

LightGBM основан на градиентном бустинге (Gradient Boosting), где:

- Функция потерь минимизируется с помощью градиентного спуска;
- Каждое новое дерево $h_t(x)$ обучается на антиградиенте (градиенте, взятом со знаком минус), а конкретно на остатках ошибок предыдущей модели:

$$F_t(x) = F_{t-1}(x) + \eta \cdot h_t(x), \quad (2.6)$$

Где η — Шаг обучения.

Градиентный бустинг обладает следующими преимуществами:

1. Высокая точность — Один из самых сильных алгоритмов для задач регрессии и классификации. Хорошо работает с разнородными данными (числовые, категориальные признаки).
2. Гибкость — Поддерживает разные функции потерь (MSE, LogLoss, Huber и другие.
3. Автоматическая обработка нелинейностей и взаимодействий признаков — За счет того, что бустинг основан на деревьях, он хорошо выявляет нелинейные зависимости.
4. Устойчивость к переобучению — Ансамбли, основанные на деревьях в целом не склоны к переобучению за счет большого количества слабых моделей, которые могут корректировать друг друга.
5. Высокая интерпретируемость.

Среди недостатков самым явным является относительная трудоемкость обучения по сравнению с другими алгоритмами машинного обучения.

На Листинге 2.2 представлен код реализации модели градиентного бустинга для решения задачи определения тональности текста.

Листинг 2.2 — Обучение и оценка модели градиентного бустинга

```
import lightgbm

lgm_model = lightgbm.LGBMClassifier(
    n_estimators=1000,
    learning_rate=0.05,
    class_weight='balanced',
    random_state=42,
    force_row_wise=True,  # Устраняет задержку на тестирование
    max_bin=255,          # Уменьшите если нужно ускорить (по умолчанию 255)
    feature_fraction=0.8, # Используйте только 80% признаков на каждое дерево
    bagging_fraction=0.8, # Stochastic bagging
    num_threads=4,        # Явно укажите количество потоков
    verbose=1             # Отображение этапов обучения модели

)

start_time = time.time()  # Засекаем начальное время

lgm_model.fit(X_train_tfidf, y_train)

end_time = time.time()  # Засекаем конечное время
execution_time = end_time - start_time

print(f"Код выполнился за {execution_time:.4f} секунд")
model_evaluate(lgm_model, X_test_tfidf, y_test, ['Нейтральный', 'Позитивный',
                            'Негативный'], png_name='Матрица ошибок бустинг')
```

Обучение модели градиентного бустинга заняло 2346.2781 секунд или около 39 минут.

Далее на Рисунках 2.3-2.4 отображены отчет классификации и матрица ошибок для данной модели.

	precision	recall	f1-score	support
Нейтральный	0.59	0.80	0.68	5488
Позитивный	0.87	0.69	0.77	10079
Негативный	0.79	0.81	0.80	5531
accuracy			0.75	21098
macro avg	0.75	0.77	0.75	21098
weighted avg	0.78	0.75	0.76	21098

Рисунок 2.3 — Отчет о классификации для градиентного бустинга

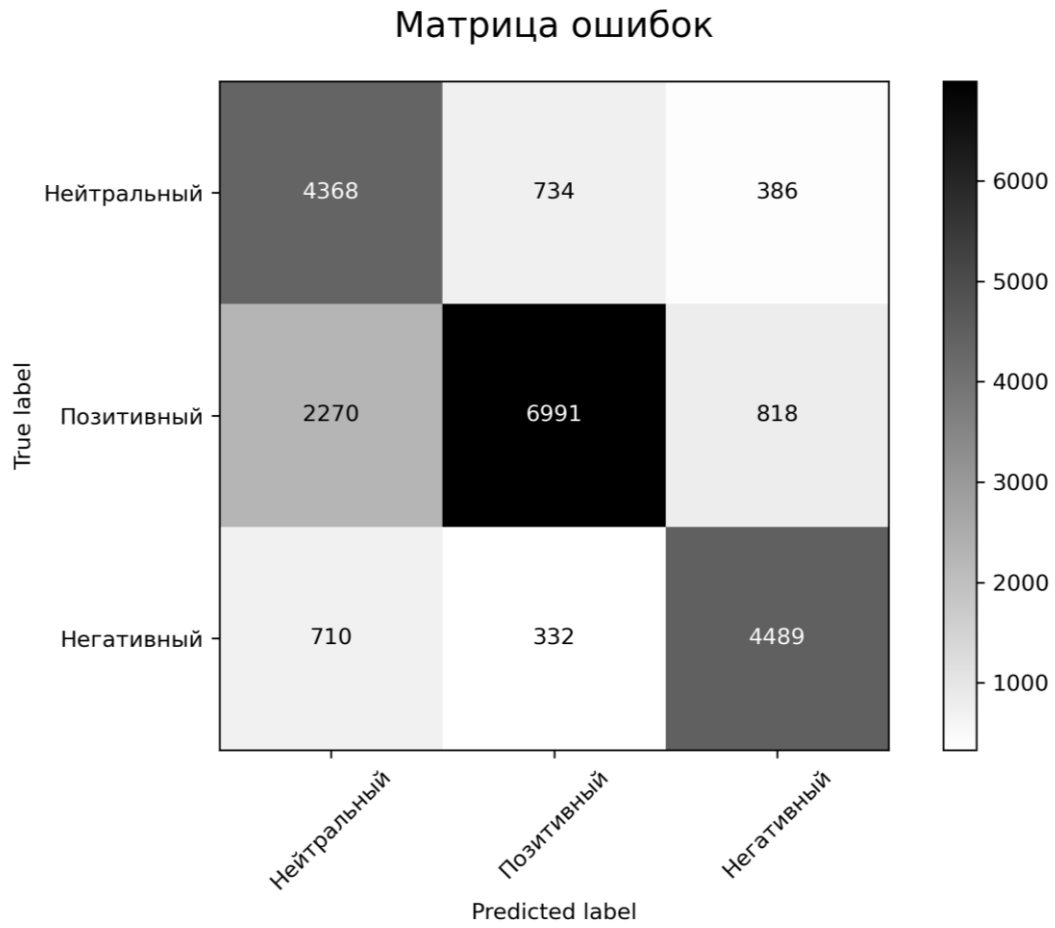

Рисунок 2.4 — Матрица ошибок для градиентного бустинга

Модель все так же определяет Нейтральный класс хуже остальных, однако по **macro avg** и **weighted avg** метрикам, в целом модель точнее, чем предыдущая.

2.1.3 SVM (Support Vector Machine)

Еще одним мощным алгоритмом классического машинного обучения является SVM (Support Vector Machine) или Метод опорных векторов. Данный алгоритм основан на поиске оптимальной разделяющей гиперплоскости в пространстве признаков.

Не углубляясь в сложную математику, SVM работает следующим образом:

1. Линейный случай:

- находит гиперплоскость, которая максимизирует зазор (margin) между классами;
- опорные вектора (support vectors) — это ближайшие точки к разделяющей границе.

2. Нелинейный случай (с ядрами — Kernel Trick):

- если данные нелинейно разделимы, SVM использует **ядерные функции** (RBF, полиномиальное и др.), чтобы преобразовать данные в пространство большей размерности, где они становятся линейно разделимыми;
- это очень элегантный способ решения задачи, так как такой способ экономит вычислительные ресурсы. Это связанно с тем, что фактически не происходит перерасчета всех признаков в новое пространство, а лишь считается «похожесть» исходного и целевого пространства.

Плюсы SVM:

1. Эффективность в высокоразмерных пространствах:

- хорошо работает, когда число признаков больше числа объектов (например, тексты, гены).

2. Универсальность за счет ядер (Kernel Trick):

- Может решать сложные нелинейные задачи с помощью RBF, полиномиального и других ядер.

Минусы:

1. Плохая масштабируемость на большие данные
2. Чувствительность к шуму и несбалансированным классам

В Листинге 2.3 представлен код реализации SVC (Support Vector Classifier) для решения задачи определения тональности текста.

Листинг 2.3 — Обучение и оценка модели классификатора на основе опорных векторов

```
from sklearn.svm import LinearSVC


# Определение параметров модели
SVC_model = LinearSVC(
    class_weight='balanced', # Учёт дисбаланса классов
    random_state=42,
    verbose=1
)

start_time = time.time()  # Засекаем начальное время

SVC_model.fit(X_train_tfidf, y_train)
end_time = time.time()  # Засекаем конечное время
execution_time = end_time - start_time
print(f"Код выполнился за {execution_time:.4f} секунд")

model_evaluate(SVC_model, X_test_tfidf, y_test, ['Нейтральный', 'Позитивный',
                                  'Негативный'], png_name='Матрица ошибок SVM')
```

Обучение заняло 38.5341 секунд. На Рисунках 2.5-2.6 представлены все соответствующих метрики для модели на основе метода опорных веткоров.

Отчёт о классификации:

	precision	recall	f1-score	support
Нейтральный	0.61	0.69	0.64	5488
Позитивный	0.82	0.76	0.79	10079
Негативный	0.78	0.79	0.79	5531
accuracy			0.75	21098
macro avg	0.74	0.74	0.74	21098
weighted avg	0.75	0.75	0.75	21098

Рисунок 2.5 — Отчет о классификации для метода опорных векторов

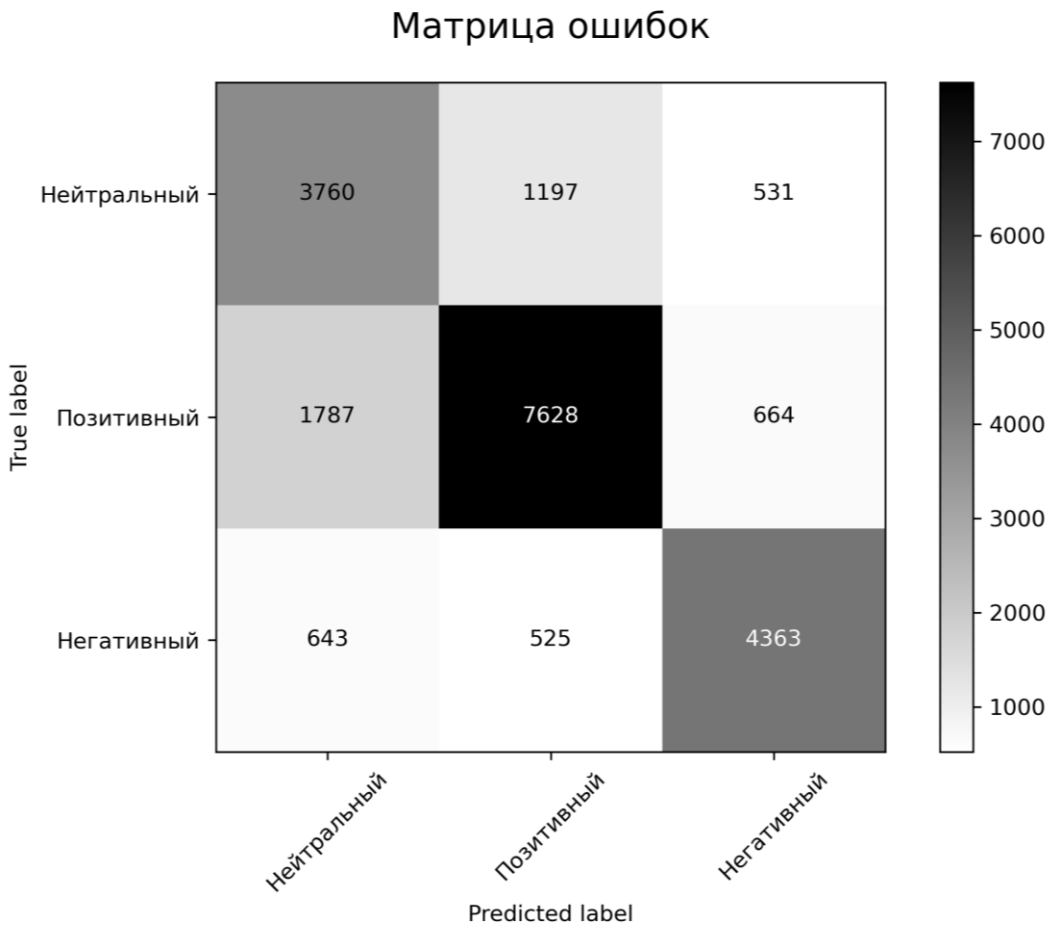

Рисунок 2.6 — Матрица ошибок для метода опорных векторов

Как видно из значений метрик, метод опорных векторов с линейным ядром (лучший выбор для работы с текстовыми данными) наилучшим образом определяет Нейтральный класс, однако чуть проигрывает по точности определения других классов предыдущим моделям. Учитывая, что модель обучилась довольно быстро и ее потенциал для более точного разделения

классов, как видно выше, чем у остальных моделей, после более тщательного подбора гиперпараметров могла бы быть наиболее оптимальным вариантов среди моделей классического машиного обучения.

Таким образом было произведено обучение и оценивание самых известных моделей классического машинного обучения, были сравнены их скорости выполнения, а также точности их предсказаний.

2.2 Нейросетевые методы

Нейросетевые методы для решения задачи определения тональности текста отличаются значительным расширением возможностей по сравнению с классическими методами на основе BoW и классических моделей машинного обучения. Они отличаются тонкий понимаем взаимосвязей между словами и даже предложения (Механизм self-attention в трансформерах и методы сверток и пулинга в сверточных архитектурах нейронных сетей).

За счет совокупности наличия большего количества параметров (а значит и более тонкого понимания разделения классов) и механизмов для определения контекста и связей между слов, данные подходы отличаются особым качеством и точностью предсказания.

2.2.1 BERT (Bidirectional Encoder Representations from Transformers)

BERT — Это большая языковая модель от Google, основанная на трансформерах и предназначенная для глубокого понимания контекста в тексте. В отличие от предыдущих моделей, BERT читает текст в обе стороны: слева-направо и справа-налево.

BERT базируется на **экодере трансформера**, предложенном в оригинальной статье. На вход ему подаются токены (слова, субсловные

единицы, буквы), которые преобразуются в векторы (эмбеддинги). К этому вектору затем прибавляются другие виды эмбеддингов, такие как позиционные эмбеддинги и сегментные эмбеддинги.

Ключевой идеей является то, что каждый токен «смотрит» на другие токены через механизм внимания (self-attention).

То есть для каждого вектора x вычисляются запрос Q, ключ K и значение V:

$$Q = xW^Q, K = xW^K, V = xW^V, \quad (2.7)$$

Где:

- x — векторное представление токена;
- W^Q— матрица весов для вычисления вектора запроса;
- W^K — матрица весов для вычисления вектора ключа;
- W^V — матрица весов для вычисления веткора значений.

Attention между токенами:

$$Attention(Q, K, V) = Concat(head_1, \dots, head_n)W^O, \quad (2.8)$$

Где:

- W^O — матрица весов, специфичная для вычисления внимания;
- $head_n$ — конкретная головка внимания.

Разные головки могут улавливать разные связи между словами и предложениями, поэтому обычно их несколько для более глубокого понимания и анализа текста.

Соответственно каждый encoder-слой имеет примерно следующий вид:

$$FFN(x) = ReLU(xW_1 + b_1)W_2 + b_2, \quad (2.9)$$

Где:

- $ReLU(x)$ — функция активации;
- W_n, b_n — соответствующие коэффициенты.

BERT-base содержит 12 таких блоков (для каждой позиции токена — одно и то же количество слоёв).

Среди преимуществ BERT можно выделить:

1. Двунаправленность — изучает текст в оба направления и слева, и справа, в отличие от тех же LSTM или GPT.
2. Заранее предобучена на огромном корпусе слов (Wikipedia, BookCorpus), что дает богатые языковые представления.
3. Высокая точность.

Среди минусов можно выделить большое количество параметров для fine-tuning (около 150 миллионов), долгое время обучения и медленную работу.

В Листинге 2.4 представлен код для реализации обучения BERT-нейронной сети для выполнения задачи определения тональности текста.

Листинг 2.4 — Обучение и оценка модели BERT

```
import pandas as pd
from datasets import Dataset
from sklearn.model_selection import train_test_split

df = pd.read_csv("df_datasets_prep.csv")
df = df.dropna(subset=["text"])
df["sentiment"] = df["sentiment"].astype(int)

train_df,       test_df       =       train_test_split(df,       test_size=0.1,
stratify=df["sentiment"], random_state=42)
train_dataset = Dataset.from_pandas(train_df.reset_index(drop=True))
test_dataset = Dataset.from_pandas(test_df.reset_index(drop=True))

from transformers import AutoTokenizer

checkpoint = "DeepPavlov/rubert-base-cased"
tokenizer = AutoTokenizer.from_pretrained(checkpoint)

def tokenize_function(example):
    return tokenizer(example["text"], truncation=True)

train_dataset = train_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)

      from transformers import AutoModelForSequenceClassification
```

Продолжение Листинга 2.4

```
model = AutoModelForSequenceClassification.from_pretrained(checkpoint,
num_labels=3)

from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="./results",
    evaluation_strategy="epoch",
    save_strategy="epoch",
    num_train_epochs=3,
    per_device_train_batch_size=4,          # малый batch под слабую GPU
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=4,
    warmup_steps=50,
    learning_rate=2e-5,
    weight_decay=0.01,
    logging_dir="./logs",
    logging_steps=50,
    save_total_limit=1,
    load_best_model_at_end=True,
    metric_for_best_model="f1_macro",
    dataloader_num_workers=0,
    fp16=False
)

from sklearn.metrics import accuracy_score, f1_score
from transformers import Trainer, DataCollatorWithPadding
import numpy as np

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)
    return {
        "accuracy": accuracy_score(labels, preds),
        "f1_macro": f1_score(labels, preds, average="macro"),
    }

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
```

Продолжение Листинга 2.4

```
    eval_dataset=test_dataset,
    tokenizer=tokenizer,
    data_collator=DataCollatorWithPadding(tokenizer),
    compute_metrics=compute_metrics,
)
start_time = time.time()  # Засекаем начальное время
trainer.train()
end_time = time.time()  # Засекаем конечное время
execution_time = end_time - start_time

print(f"Код выполнился за {execution_time:.4f} секунд")



# Оценка модели
      predictions = trainer.predict(test_dataset)
      y_true = predictions.label_ids
      y_pred = np.argmax(predictions.predictions, axis=1)

      # Classification report
      print(classification_report(y_true, y_pred, target_names=["Нейтральный",
"Позитивный", "Негативный"]))

      # Confusion matrix
      conf_matrix = confusion_matrix(y_true, y_pred)

      # Визуализация
      plt.figure(figsize=(6, 5))
      sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="binary",
                  xticklabels=["Нейтральный", "Позитивный", "Негативный"],
                  yticklabels=["Нейтральный", "Позитивный", "Негативный"])
      plt.xlabel("Предсказано")
      plt.ylabel("Истинно")
      plt.title("Матрица ошибок")
      plt.tight_layout()
plt.show()
```

Обучение заняло 112 872 секунды или 31 час 21 минуту и 12 секунд. Далее на Рисунках 2.7-2.8 приведены результаты оценки качества модели.

Отчёт о классификации:

	precision	recall	f1-score	support
Нейтральный	0.88	0.89	0.88	5488
Позитивный	0.92	0.88	0.90	10079
Негативный	0.85	0.91	0.88	5531
accuracy			0.89	21098
macro avg	0.88	0.89	0.89	21098
weighted avg	0.90	0.88	0.90	21098

Рисунок 2.7 — Отчет о классификации для BERT-модели

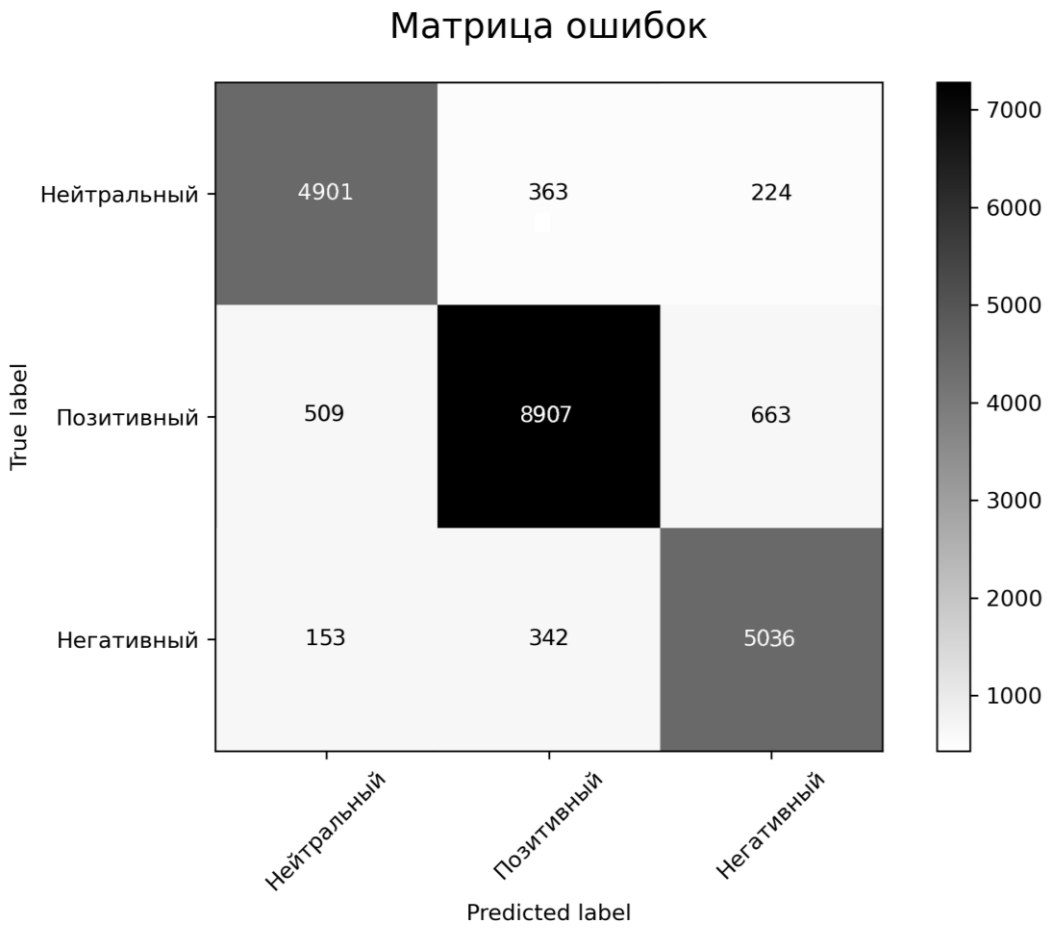

Рисунок 2.8 — Матрица ошибок для BERT-модели

Как и следовало ожидать, нейросетевой подход с использованием BERT-модели демонстрирует крайне высокую точность в предсказании всех классов, даже нейтрального.

2.2.2 Yoon Kim CNN

Yoon Kim CNN — это сверточная нейросеть для обработки текста, впервые представлена в статье **"Convolutional Neural Networks for Sentence Classification" (2014)**, автор — *Yoon Kim* (NYU). Эта модель стала одной из первых и наиболее влиятельных реализаций CNN в задаче классификации текста (например, по тональности).

Хотя CNN изначально разрабатывались для обработки изображений, Yoon Kim показал, что они эффективно работают и для анализа предложений. Основная идея — применять **свертки (filters)** к матрице векторных представлений слов, чтобы выявить локальные (n-граммные) паттерны, важные для классификации текста.

Предложение длинной n слов представляется как матрица

$$S \in R^{n \times d}, \tag{2.10}$$

Где:

- n — длина предложения;
- d — размерность эмбеддинга слова (например, GloVe, Word2Vec).

Используется фильры размера $h \times d$, где h — количество слов в n-грамме. Фильтр $\boldsymbol{w} \in R^{h \times d}$ применяется ко всем возможным подматрицам $S_{i:i+h-1}$, давая скаляр:

$$\mathrm{c}_i = f(\langle \boldsymbol{w}, S_{i:i+h-1} \rangle + b) \tag{2.11}$$

Где:

- $\langle .,. \rangle$ — скалярное произведение (или элемент-wise произведение и сумма);
- f — функция активации (обычно ReLU или tanh);
- b — смещение (bias).

Далее применяется **max-overtime-pooling**:

$$\hat{\mathrm{c}} = max\{c_1, c_2, \dots, c_{n-h+1}\} \tag{2.12}$$

Это фиксирует наиболее "выразительное" срабатывание фильтра.

Yoon Kim CNN обладает следующими преимуществами:

1. Простота и скорость — очень лёгкая и быстрая модель по сравнению с RNN или трансформерами.
2. Хорошие результаты — несмотря на простоту, даёт отличные результаты на задачах вроде SST-2, TREC, MR и др.
3. Локальность — эффективно выявляет n-граммные признаки (например, «не хорошо», «очень плохо»).
4. Восприимчива к обучаемым embedding — можно использовать как статические (GloVe), так и обновляемые в процессе обучения.

Минусы:

1. Игнорирует порядок вне окна — модель «не знает», где именно встретился фильтр, она использует только максимальное срабатывание.
2. Не работает на длинных зависимостях — неспособна уловить зависимости между далеко расположенными словами.

Ниже на Листинге 2.5 представлен код для обучения и оценивания модели Yoon Kim CNN.

Листинг 2.5 — Обучение и оценка модели Yoon Kim CNN

```
# Импорт библиотек и установка гиперпараметров
import pandas as pd
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn
import torch.nn.functional as F
from sklearn.model_selection import train_test_split
from torchtext.vocab import build_vocab_from_iterator
from torchtext.data.utils import get_tokenizer
from tqdm import tqdm
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Гиперпараметры
MAX_LEN = 100
EMBED_DIM = 128
BATCH_SIZE = 64
NUM_CLASSES = 3
EPOCHS = 5
FILTER_SIZES = [3, 4, 5]
NUM_FILTERS = 100
LR = 1e-3
```

Продолжение Листинга 2.5

```
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Загрузка и подготовка датасета: токенизация, векторизация, паддинг
df = pd.read_csv("df_datasets_prep.csv").dropna(subset=["text"])
df["sentiment"] = df["sentiment"].astype(int)

train_texts, test_texts, train_labels, test_labels = train_test_split(
            df["text"].tolist(), df["sentiment"].tolist(), test_size=0.1,
            stratify=df["sentiment"], random_state=42
)

tokenizer = get_tokenizer("basic_english")

def yield_tokens(data):
    for text in data:
        yield tokenizer(text)

# Построение словаря токенов
vocab = build_vocab_from_iterator(yield_tokens(train_texts), specials=["<pad>",
"<unk>"])
vocab.set_default_index(vocab["<unk>"])

# Функция кодирования текста
def encode(text):
    tokens = tokenizer(text)
    ids = vocab(tokens)
    if len(ids) < MAX_LEN:
        ids += [vocab["<pad>"]] * (MAX_LEN - len(ids))
    else:
        ids = ids[:MAX_LEN]
    return ids

# Класс Dataset для PyTorch
class TextDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = [torch.tensor(encode(t)) for t in texts]
        self.labels = torch.tensor(labels)

    def __len__(self): return len(self.labels)
    def __getitem__(self, idx): return self.texts[idx], self.labels[idx]

train_ds = TextDataset(train_texts, train_labels)
test_ds = TextDataset(test_texts, test_labels)

train_loader = DataLoader(train_ds, batch_size=BATCH_SIZE, shuffle=True)
test_loader = DataLoader(test_ds, batch_size=BATCH_SIZE
      # Определение архитектуры модели Kim Yoon CNN
      class KimCNN(nn.Module):
          def __init__(self, vocab_size, embed_dim, num_classes, filter_sizes,
num_filters):
              super(KimCNN, self).__init__()
              self.embedding     =     nn.Embedding(vocab_size,     embed_dim,
padding_idx=vocab["<pad>"])
              self.convs = nn.ModuleList([
                  nn.Conv2d(1,   num_filters,   (fs,   embed_dim))   for   fs   in
filter_sizes
              ])
              self.dropout = nn.Dropout(0.5)
              self.fc = nn.Linear(num_filters * len(filter_sizes), num_classes)

          def forward(self, x):
              x = self.embedding(x)  # [B, L, D]
```

Продолжение Листинга 2.5

```
            x = x.unsqueeze(1)      # [B, 1, L, D]
            x = [F.relu(conv(x)).squeeze(3) for conv in self.convs]  # [B, F,
L']
            x = [F.max_pool1d(i, i.size(2)).squeeze(2) for i in x]   # [B, F]
            x = torch.cat(x, dim=1)  # [B, F * len(filter_sizes)]
            x = self.dropout(x)
            return self.fc(x)

model    =    KimCNN(len(vocab),    EMBED_DIM,    NUM_CLASSES,    FILTER_SIZES,
NUM_FILTERS).to(DEVICE)
      # Обучение модели с логированием потерь и точности
      optimizer = torch.optim.Adam(model.parameters(), lr=LR)
      criterion = nn.CrossEntropyLoss()
start_time = time.time()  # Засекаем начальное время
      for epoch in range(EPOCHS):
          model.train()
          total_loss, total_correct = 0, 0
          for    inputs,    labels    in    tqdm(train_loader,    desc=f"Epoch
{epoch+1}/{EPOCHS}"):
              inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)
              optimizer.zero_grad()
              outputs = model(inputs)
              loss = criterion(outputs, labels)
              loss.backward()
              optimizer.step()
              total_loss += loss.item() * inputs.size(0)
              total_correct += (outputs.argmax(1) == labels).sum().item()
          acc = total_correct / len(train_loader.dataset)
    print(f"Train Loss: {total_loss / len(train_loader.dataset):.4f}, Accuracy:
{acc:.4f}")
end_time = time.time()  # Засекаем конечное время
execution_time = end_time - start_time
print(f"Код выполнился за {execution_time:.4f} секунд")

      # Оценка модели: отчёт по классам и матрица ошибок
      model.eval()
      all_preds = []
      all_labels = []

      with torch.no_grad():
          for inputs, labels in test_loader:
              inputs, labels = inputs.to(DEVICE), labels.to(DEVICE)
              outputs = model(inputs)
              preds = outputs.argmax(1)
```

Продолжение Листинга 2.5

```
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Текстовый отчёт
    print("Classification Report:")
    print(classification_report(all_labels, all_preds,
target_names=["Нейтральный", "Позитивный", "Негативный"]))

    # Матрица ошибок
    cm = confusion_matrix(all_labels, all_preds)
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt="d", cmap="binary",
                xticklabels=["Нейтральный", "Позитивный", "Негативный"],
                yticklabels=["Нейтральный", "Позитивный", "Негативный"])
    plt.xlabel("Предсказано")
    plt.ylabel("Истинно")
    plt.title("Матрица ошибок")
    plt.tight_layout()
plt.show()
```

Обучение модели заняло 8 803 секунды или примерно 2 часа и 26 минут и 43 секунды. На Рисунках 2.9-2.10 представлены демонстрации всех соответствующих метрик.

Отчёт о классификации:

	precision	recall	f1-score	support
Нейтральный	0.72	0.73	0.72	5488
Позитивный	0.87	0.87	0.87	10079
Негативный	0.77	0.77	0.77	5531
accuracy			0.80	21098
macro avg	0.79	0.79	0.79	21098
weighted avg	0.80	0.78	0.79	21098

Рисунок 2.9 — Отчет о классификации для Yoon Kim CNN

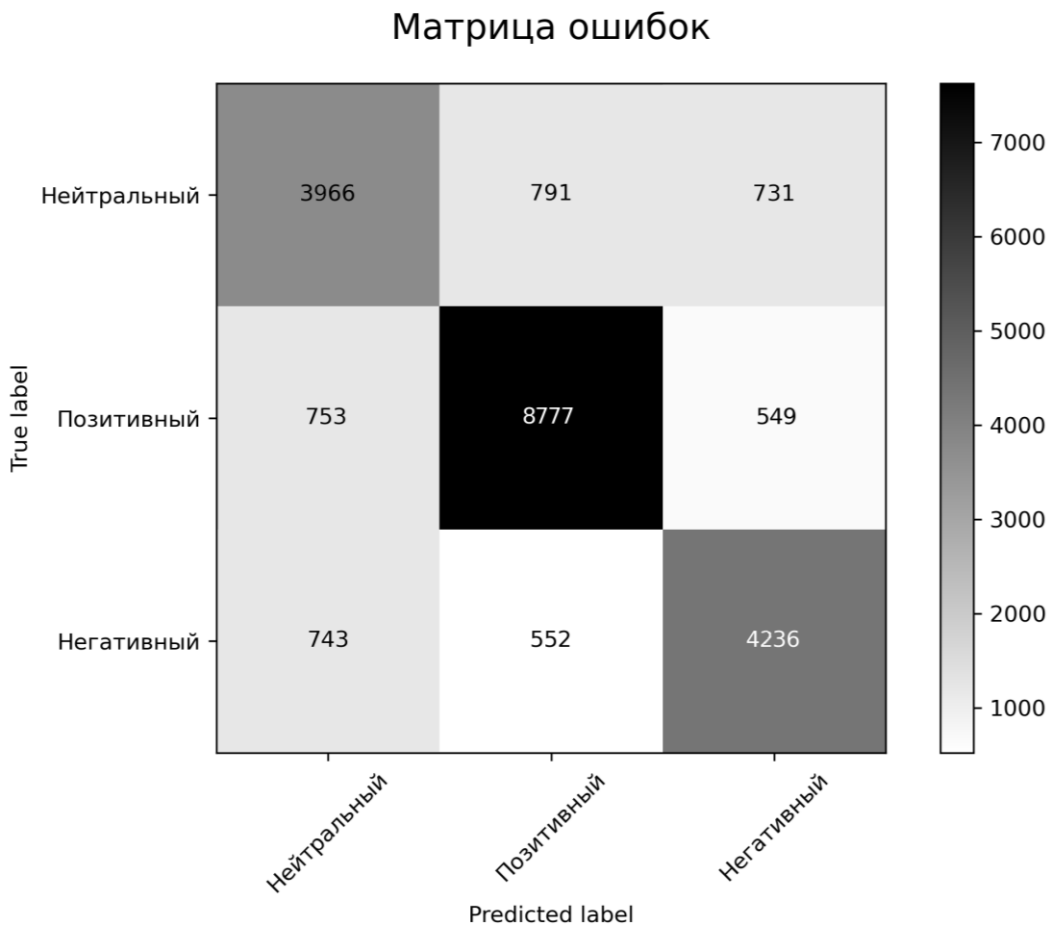

Рисунок 2.10 — Матрица ошибок для Yoon Kim CNN

Сверточная нейронная сеть показывает на порядок более слабые результаты, чем трансформер, однако она все еще лучше стандартных алгоритмов машинного обучения, также можно заметить довольно хорошие метрики для Нейтрального класса (выше 0.7). К тому же, стоит брать во внимание, что обучение произвелось за довольно вменяемое количество времени, чего нельзя утверждать о BERT-модели.

2.3 Анализ результатов

После обучения всех представленных ранее классификаторов нужно подвести общие итоги и дать комментарии по каждому типу моделей.

В Таблице 2.2 будут представлены данные об моделях, о времени их обучения, типе обучения и значению ключевых метрик оценки качества модели.

Таблица 2.2 — Итоги обучения

Название модели	**Время обучения (секунд)**	**Поддержка GPU**	**Среднее F1-score**	**Accuracy**
Логистическая регрессия	17,824	Нет	0,74	0,74
Градиентный бустинг	2346,2781	Нет	0,75	0,75
Метод опорных векторов	38,5341	Нет	0,75	0,74
BERT	112872,8573	Да	0,89	0,89
Yoon Kim CNN	8803,3325	Да	0,80	0,79

Из таблицы видно, что классические модели машинного обучения на порядок хуже по качеству и предсказательным способностям, чем нейросетевые подходы, однако обучаются они чаще всего за ничтожно малое количество времени, при этом получая на выходе неплохие показатели метрик.

Нейросетевые подходы, как ни странно, демонстрируют гораздо лучшее качество и обобщающие свойства, чем предшествующие модели. Они без труда классифицируют Нейтральный класс, что с трудом удавалось методам классического машинного обучения. Однако, как видно, колоссальный недостаток данных подходов в огромном времени, которое требуется на их обучение, особенно в случае BERT-модели.

Таким образом, можно сделать вывод, что если требуется быстрое решение с некритическими требованиями по точности предсказания — смело можно использовать методы классического машинного обучения.

Если необходима очень точная модель, то это безусловно BERT, однако если требуется баланс между сложностью и точностью, неплохим выбором будет Yoon Kim CNN.

ЗАКЛЮЧЕНИЕ

В ходе проведённого исследования была проанализирована эффективность различных подходов к задаче определения тональности текста — как классических методов машинного обучения, так и современных нейросетевых архитектур. Основное внимание было уделено сравнению моделей по ключевым критериям: точности классификации, скорости обучения, ресурсоёмкости, зависимости от объема обучающих данных и степени интерпретируемости.

В результате экспериментов было подтверждено, что нейросетевые модели, в частности архитектура BERT, значительно превосходят классические методы (например, логистическую регрессию или SVM) по качеству классификации, особенно при наличии большого объема обучающих данных. Однако они требуют значительно больше вычислительных ресурсов и времени на обучение, а также характеризуются низкой интерпретируемостью из-за своей «чёрной ящичной» природы.

С другой стороны, классические модели демонстрируют достойные результаты при ограниченных ресурсах и обладают высокой скоростью обучения и простой структурой, что делает их удобными для быстрого прототипирования и применения в системах с жёсткими ограничениями по времени или инфраструктуре.

Модель Kim Yoon CNN показала себя как компромиссное решение — она обеспечивает хорошее качество при умеренной ресурсоёмкости, уступая BERT по метрикам, но выигрывая по скорости и простоте реализации.

Таким образом, выбор метода решения задачи определения тональности текста должен основываться на балансе между требуемым качеством, доступными ресурсами и условиями применения. В рамках данной работы был получен ценный практический и теоретический опыт, позволяющий осознанно подходить к выбору подходящих моделей для задач обработки естественного языка.

СПИСОК ЛИТЕРАТУРЫ

1. Морфологический анализатор Pymorphy2 [Электронный ресурс]. — 2025. — URL: https://habr.com/ru/companies/trinion/articles/332772/ (дата обращения: 15.05.2025).
2. NLTK Documentation [Электронный ресурс]. — 2025. — URL: https://pymorphy2.readthedocs.io/en/stable/user/guide.html (дата обращения: 16.05.2025).
3. Documentation GENSIM topic modeling for humans [Электронный ресурс]. — 2025. — URL: https://radimrehurek.com/gensim/auto_examples/index.html (дата обращения: 20.05.2025).
4. Logistic Regression [Электронный ресурс]. — 2025. — URL: https://scikitlearn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html (дата обращения: 5.06.2025).
5. Ансамбли в машинном обучении [Электронный ресурс]. — 2025. — URL: https://education.yandex.ru/handbook/ml/article/ansambli-v-mashinnom-obuchenii (дата обращения: 7.06.2025).
6. Краткий обзор алгоритма машинного обучения Метод Опорных Векторов (SVM) [Электронный ресурс]. — 2025. — URL: https://habr.com/ru/articles/428503/ (дата обращения 10.06.2025).
7. Объясняем простым языком что такое трансформеры [Электронный ресурс]. — 2025. — URL: https://habr.com/ru/companies/mws/articles/770202/ (дата обращения 12.05.2025).
8. Себастьян Рашка. Python и машинное обучение. — 2-е изд. — Москва: ДМК Пресс, 2017. — 420с.

9. CNN for Sentence Classification by Yoon Kim [Электронный ресурс]. — 2025. — URL: https://www.kaggle.com/code/hamishdickson/cnn-for-sentence-classification-by-yoon-kim (дата обращения 15.06.2025).