МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

АВТОМАТИЧЕСКАЯ ТЕМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ НОВОСТНОГО МАССИВА

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Кондрашова Даниила Владиславовича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Папшев

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время обработка больших объёмов текстовых данных, включая новостные потоки, становится критически важной задачей. Как в научной среде, так и в бизнесе требуется оперативно анализировать информацию, отслеживать тенденции и принимать решения. Однако анализ всего массива данных невозможен из-за его масштабов, необходимо фильтровать информацию, оставляя только нужную.

Помочь в решении этой проблемы может тематическая классификация. Хотя многие сайты и порталы предлагают рубрикацию контента, её точность часто оказывается низкой: теги присваиваются некорректно или поверхностно. Это приводит к ошибкам в поиске и анализе информации.

В таком случае необходим механизм позволяющий получать правильную тематическую классификацию данных, который смог бы присваивать темы тем же новостям автоматически. Одни из возможных инструментов, которые позволяют реализовать подобие такого механизма — это тематические модели и алгоритмы машинного и глубокого обучения. Первый из них позволяет косвенно выявить темы текстового набора данных и разметить данные для обучения второго инструмента, который сможет тематически классифицировать последующий текст.

Таким образом, целью данной работы является реализация механизма автоматической тематической классификации новостей с помощью методов тематического моделирования и глубокого и машинного обучения.

Для достижения этой цели необходимо решить следующие задачи:

1. Реализовать механизм получения новостных массивов данных;
2. Реализовать механизм подготовки текстовых данных;
3. Вычислить тематические модели;
4. Путём сравнительного анализа выявить наиболее удачную тематическую модель;
5. Разметить данные для обучения на них моделей машинного и глубокого обучения;
6. Обучить несколько моделей машинного и глубокого обучения и выявить наиболее удачную путём сравнительного анализа;
7. Провести анализ получившихся результатов.

1 Теоретические и методологические основы

1.1 Получение текстовых данных

1.1.1 Выбор инструмента

Для получения каких-либо данных с сайта существует три основных метода:

— Ручной метод — выписывание необходимой информации с помощью человека;
— Получение данных путём предоставления их запроса у владельца, с их последующим скачиванием;
— Получение данных программным путём.

Первый метод из-за своей неэффективности можно сразу отбросить. Второй метод далеко не всегда можно применить, кроме того вряд ли владельцы информационных платформ будут оперативно отсылать все данные по первой просьбе. Таким образом, остаётся только третий метод.

Оперативно и достаточно эффективно в большинстве случаев можно получить данные применяя инструменты вебскраппинга. Дальше будет использоваться этот вариант получения новостного массива.

Различные библиотеки для вебскраппинга доступны на разных языках, однако исходя из того, что наиболее популярным языком для обработки данных и работы с машинным и глубоким обучением является python, выберем библиотеки доступные на нём. Такими библиотеками ялвяются requests, beautifulsoap4 и selenium. Первая бибилиотека позволяет отсылать http запросы. Вторая библиотека позволяет преобразовывать html код в подобие классов для удобного получения информации. Последняя библиотека позволяет обрабатывать сайта, которые по http запросу не выдают html код наблюдаемой пользователем страницы. Данная библиотека позволяет эмулировать работу браузера и получать html код страницы прямо из него.

Такого набора хватит для обработки подавляющего большинства сайтов.

1.1.2 Подбор информационной платформы

В рамках данной работы среди всех типов текстовых данных будут рассматриваться новостные. Теперь нужно подобрать сайт.

Если для получения информации есть несоклько возможных веб-источников, то стоит выбирать сайт по следующим критериям:

1. Сайт имеет единую структуру документов;
2. Сайт не блокирует http запросы отправляемые вебскраппером;
3. Сайт не является реактивным, то есть в момент просмотра страницы html код страницы полностью сформирован и доступен по запросу клиенту.

Будет идеально, если все пункты соблюдаются, одако, даже в случае отсутствия пунктов 2 и 3, ограничения в большинстве случаев можно достаточно просто обойти. В случае несоответствия пункта 1 могут возникнуть серьёзные трудности, которые, в худшем случае, решить только методами веб скраппинга не получится.

В рамках данной работы будет использоваться новостной сайт ВШЭ. Данный сайт соответствует всем описанным выше критериям.

1.2 Подготовка текстовых данных

Полученные данные требуют предварительной обработки для устранения шума и повышения качества анализа. Основные этапы включают:

1. Очистка от технического шума:
   - Удаление лишних пробелов, переносов строк;
   - Очистка от спецсимволов (скобки, HTML-теги, эмодзи);
   - Нормализация регистра (приведение всего текста к нижнему регистру).
2. Токенизация: Разделение текста на слова или предложения;
3. Лемматизация: Приведение слов к начальной форме (например, «бежал» $\Rightarrow$ «бежать»);
4. Удаление стоп-слов: Исключение частых слов без смысловой нагрузки (предлоги, частицы, местоимения);

Обоснование выбора лемматизации вместо стемминга: Стемминг (например, алгоритм Snowball) «обрубает» окончания по шаблонам («бежал» $\Rightarrow$ «беж»), что искажает смысл. Лемматизация сохраняет семантику, что критично для тематического моделирования.

1.2.1 Выбор инструментов

Чтобы не повышать количество используемых языков, будем рассматривать только инструменты, доступные на Python. Среди них выделяются: NLTK, Pymorphy3, SpaCy и Gensim.

Сделаем выбор между связкой NLTK + Pymorphy3 и SpaCy. Обе группы

библиотек позволяют проводить лемматизацию и удаление стоп-слов, но реализуют это по-разному. NLTK и Pymorphy3 приводят слова к начальной форме без учёта контекста, тогда как SpaCy — нейросетевой инструмент, анализирующий окружение терминов. Определение стоп-слов в обоих случаях происходит по заранее заданным словарям, поэтому разницы здесь нет. Однако SpaCy обеспечивает не только более точную лемматизацию, но и лаконичный интерфейс, что упрощает интеграцию в проект.

Как упоминалось ранее библиотека SpaCy определяет стоп-слова только по предопределённому списку, который не является исчерпывающим. Это связано с тем, что набор стоп-слов зависит от тематики текста, и универсального решения не существует. Для дополнительной фильтрации применим метрику TF-IDF, которая оценивает значимость слов. Формула расчёта:

$$tfidf(w,d) = \frac{n_{wd}}{n_d} \cdot \log\left(\frac{|D|}{|\{d \in D : w \in d\}|}\right), \tag{1}$$

где:

— w — термин;
— d — документ;
— n_{wd} — частота встречаемости w в d;
— n_d — число терминов в d;
— $|D|$ — число документов в коллекции;
— $|\{d \in D : w \in d\}|$ — количество документов, содержащих w.

Данная метрика будет тем выше для термина w в документе d, чем чаще будет встречаться термин w в документе d и реже во всех остальных документах коллекции. Таким образом, данную метрику можно интерпретировать как метрику значимости слова w для документа d. Её расчёт будет производиться с помощью билиотеки Gensim.

Таким образом, для обработки текста выбраны SpaCy (токенизация, лемматизация, базовые стоп-слов?) и Gensim (расширенная фильтрация через TF-IDF).

2 Практико-технологические основы

2.1 Получение новостного массива путём вебскраппинга

Для обработки такого простого новостного сайта как у ВШЭ достаточно использования requests и beautifulsoap4, без selenium.

Чтобы наиболее просто и эффективно получить данные необходимо разобрать структуру сайта и разработать соответствующие функции под каждую из частей. Сам портал представляет собой многостраничный сайт, на каждой странице которого расположено по 10 новостей с краткой информацией по каждой: ссылка, дата, заголовок, краткое содержание. На каждую новость можно перейти по ссылке для получения полного её содержания.

Теперь последовательно реализуем функции-обработчики под соответствующие части сайта.

Чтобы получать html код страницы, необходимо воспользоваться библиотекой requests и методом get. Данный метод отправляет запрос на сайт и получает соответствующий код в качестве ответа, который можно сохранить в файл для последующей выгрузки и обработки. Соответствующая функция расположена в листинге 1.

```
def __getPage__(url: str, file_name: str) -> None:
    # получение html кода страницы с помощью библиотеки requests
    r = requests.get(url=url)
    # сохранение полученного кода в текстовый файл
    with open(file_name, "w", encoding="utf-8") as file:
        file.write(r.text)
```

Листинг 1: Функция получения html кода страницы

Далее нужно реализовать получение краткой информации о новости: ссылка, дата, краткое содержание. Для этого нужно загрузить код страницы из файла и преобразовать его к классам с помощью библиотеки beautifulsoap4. Далее можно будет воспользоваться поиском по тегам и классам с помощью метода find и получить текстовое содержимое с помощью методов text и get. Пример получения ссылки и краткого содержания новости можно увидеть в данном листинге 2.

```
# получение html кода страницы из файла
with open(page_file_name, encoding="utf-8") as file:
        src = file.read()
# преобразование html кода в классы
```

```
5  soup = BeautifulSoup(src, "lxml")
6  # переход к содержимому новости, которое находится
7  # в теге div с классом post
8  news = soup.find("div", class_="post")
9  try:
10     # получение текста ссылки из соответствующего тега
11     link = news.find("h2",
               class_="first_child").find("a").get("href")
12     # не все ссылки в теге сохранены полностью, данный
13     # код добавляет обрезанную часть
14     if not link.startswith("https://"):
15         link = 'https://www.hse.ru' + link
16 except:
17     link = ""
18 try:
19     # получение краткого описания новости из соответствующего тег
           a
20     news_short_content = news.find("p",
               class_="first_child").find_next_sibling("p").text.strip()
21 except:
22     news_short_content = ""
```

Листинг 2: Получение ссылки и краткого содержания

Теперь нужно реализовать функцию получения полного содержания новости. Для этого нужно воспользоваться реализованной функцией get_page (получить код страницы по полученной ранее ссылке на новость), преобразовать его в классы с помощью beautifulsoap4 и получить текстовое содержимое с помощью методов find и text. Реализацию соответствующей функции можно увидеть в листинге 3.

```
1  def __parse_news__(url: str) -> str:
2      # получаем html код страницы по ссылке на новость
3      news_file_name = "news.html"
4      __getPage__(url, news_file_name)
5      # и сразу загружаем его из файла
6      with open(news_file_name, encoding="utf-8") as file:
7          src = file.read()
8      # преобразуем html код к классам и сразу получаем всё текстов
           ое содержание
9      # новости. Это возможно так как весь контент новости содержит
           ся
```

```
10      # в теге post__text
11      content = BeautifulSoup(src, "lxml").find("div",
                class_="main").find(
12          "div", class_="post__text"
13      ).text.strip()
14      # возвращаем полученное содержание новости в виде строки
15      return content
```

Листинг 3: Функция получения полного текстового содержания новости

Следующим шагом нужно вспомнить, что на странице располагается 10 новостей, каждая новость располагается в теге div с классом post. Таким образом, нужно 10 раз проитерироваться по данным тегам и получить 10 новостей. Сделать это можно с помощью метода find_next_sibling (он ищет следующий тег, который идентичен по типу и классу предыдущему) и обычного цикла. Хранить полученное содержимое удобно в pandas DataFrame, так как с помощью него удобно обрабатывать полученные массивы данных и вычислять их колличественные характеристики. Ключевые части соответствующей функции представлены в следующем листинге 4.

```
1  def __parse_page__(page_file_name: str, news_container:
       pd.DataFrame) -> None:
2      # скрытый фрагмент получения html кода страницы
3      for i in range(10):
4          # скрытый фрагмент получения краткой информации о новости
5          try: # получение полного содержания новости
6              if link.startswith("https://www.hse.ru/news/"):
7                  news_content = __parse_news__(link)
8          except:
9              news_content = ""
10         # сохранение содержимого новости, если она не пустое
11         if len(
12             news_day + news_month + news_year + news_name +
                   news_short_content +
13             news_content
14         ) > 0:
15             news_container.loc[len(news_container.index)] = [
16                 link, news_date, news_name, news_short_content,
                       news_content]
17         # переход к следующей новости
```

```
18            news = news.find_next_sibling("div", class_="post")
```

Листинг 4: Функция обработки одной страницы новостей

Далее необходимо реализовать функцию обрабатывающую все страницы с новостями. Сделать это можно путём многократного применения описанной выше функции обработки одной новостной страницы к изменяемой ссылке страницы. Благодаря простому устройству сайта ВШЭ менять эту ссылку можно достаточно просто с помощью обычного цикла путём изменения индекса в одной части. Соответствующий код представлен в следующем листинге 5.

```
1  def __crawling_pages__(start: int, end: int, news_container:
       pd.DataFrame, num_of_thread: int) -> pd.DataFrame:
2      page_file_name = "page.html"
3      for i in range(start, end + 1):
4          try:
5              __getPage__("https://www.hse.ru/news/page{0}.html".format(i),
                   page_file_name)
6              __parse_page__(page_file_name, news_container)
7          except:
8              continue
```

Листинг 5: Функция обработки всех страниц новостей

Осталось только для ускорения получения данных с файла добавить многопоточность. Сделать это можно с помощью стандартных средств языка python, только стоит учесть, что под каждый отдельный поток нужно будет создать свой отдельный контейнер pandas DataFrame, чтобы избежать проблем с записью. Соответствующий код представлен в следующем листинге 6.

```
1  def crawling_pages(off_pc: bool, pages: int) -> None:
2      columns = ["url", "date", "title", "summary", "content"]
3      # создание контейнеров под каждый из потоков
4      news_container1 = pd.DataFrame(columns=columns)
5      news_container2 = pd.DataFrame(columns=columns)
6      # создание потоков
7      thread1 = threading.Thread(target=__crawling_pages__,
           args=(0, pages // 2, news_container1, 1))
8      thread2 = threading.Thread(target=__crawling_pages__,
           args=(pages // 2, pages, news_container2, 2))
9      # запуск потоков
10     thread1.start()
11     thread2.start()
```

```
12      # ожидание завершения работы потоков
13      thread1.join()
14      thread2.join()
15      # объединение содержимого контейнеров потоков в один
16      try:
17          news = pd.concat([news_container1, news_container2],
                    ignore_index=True)
18          news.to_excel("./news.xlsx")
19      except:
20          print("Не получилось!")
```

Листинг 6: Многопоточное получение новостей

Полный код вебскраппера можно увидеть в соответствующем приложении А.

2.2 Подготовка новостного массива

2.2.1 Удаление лишних пробелов и переносов строк

Для корректной токенизации и просто для удобства анализа текстовых данных важно удалить из них лишние пробелы и переносы строк, сделать это можно с помощью стандартных средств языка python.

Функция будет иметь следующий алгоритм:

1. Записываем в результирующую строку символы из исходной строки, пока не будет встречен символ пробела или переноса строки;
2. Добавляем к результирующей строке 1 символ пробела и прекращаем добавление символов, пока не встретим символ, отличный от пробела или переноса строки;
3. В случае, когда встретится символ не являющийся пробелом или переносом строки переходим к пункту 1. Повторяем описанные выше действия пока не будет пройдена вся исходная строка.

Реализация соответствующей функции представлена в следующем листинге 7.

```
1  def __remove_extra_spaces_and_line_breaks__(self, text: str) ->
       str:
2      processed = ""
3      if type(text) != str or len(text) == 0:
4          return ""
5      flag = True
```

```
    for symb in text:
        if flag and (symb == " " or symb == "\n"):
            processed += " "
            flag = False
        if symb != " " and symb != "\n":
            flag = True
        if flag:
            processed += symb
    return processed.strip()
```

Листинг 7: Функция удаления лишних пробелов и переносов строк

2.2.2 Разделение строк на русские и английские фрагменты

Библиотека SpaCy обрабатывает текст с помощью различных предобученных нейронных сетей, такие сети обучаются работе только на одном языке, например, только на русском или английском языке.

Текст новостей с новостного сайта ВШЭ имеет вставки на английском языке, что делает некорректным использование только одной предобученной нейронной сети. Поэтому, чтобы применять сразу два типа нейронных сетей необходимо разбивать строки на русские и английские фрагменты. Решить данную задачу можно с помощью стандартных средств языка python.

Функция будет иметь следующий алгоритм:

1. Определяем к какому алфавиту принадлежит первый буквенный символ строки и устанавливаем идентификатор в состояние соответствующее типу алфавита;
2. Записываем последовательно символы строки во временную подстроку, пока не встретим букву другого алфавита;
3. После встречи символа противоположного алфавита записываем в список кортеж вида (идентификатор алфавита, временная подстрока);
4. Очищаем временную подстроку и изменяем состояние идентификатора на противоположное. После этого повторяем описанные выше действия, пока не будет пройдена вся исходная строка.

Реализация соответствующей функции представлена в следующем листинге 8.

```
def __first_is_en__(self, cell: str) -> bool:
        index_first_en = re.search(r"[a-zA-Z]", cell)
        index_first_ru = re.search(r"[а-яА-Я]", cell)
```

```
        return True if index_first_en and (not(index_first_ru)
                or index_first_en.start() <
                index_first_ru.start()) else False
def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
    str)]:
    parts = []
    is_en = self.__first_is_en__(cell)
    part = ""
    for symb in cell:
        if is_en == (symb in string.ascii_letters) or not
                (symb.isalpha()):
            part += symb
        else:
            parts.append((is_en, part))
            part = symb
            is_en = not (is_en)
    if part:
        parts.append((is_en, part))
    return parts
```

Листинг 8: Функция разбиения строки на русские и английские фрагменты

2.2.3 Обработка двоеточий и временных меток

При вычислении тематической модели BigARTM использует символ двоеточия как служебный, поэтому наличие его в текстовых данных приведёт к возникновению ошибок.

Само двоеточие, чаще всего, используется при написании времени, данные случаи можно обработать. Другие случаи применения предусмотреть проблематично, поэтому работать функция будет следующим образом: если двоеточие располагается в шаблоне временной метки, то будем заменять её на строку time, в противном случае будем просто удалять двоеточие.

Реализация соответствующей функции представлена в следующем листинге 9.

```
def __time_processing__(self, text: str) -> str:
    if re.match(r"\d{2}:\d{2}", text):
        return "time"
    else:
        return text.replace(":", "")

```

```
def __processing_token__(self, token_lemma: str) -> str:
    return self.__time_processing__(
        self.__remove_extra_spaces_and_line_breaks__(token_lemma)
    )
```

Листинг 9: Функция обработки двоеточий и временных меток

2.2.4 Токенизация, лемматизация и удаление стоп-слов по словарю

Библиотека SpaCy имеет простой и удобный интерфейс. Для проведения токенизации, лемматизации и обнаружении стоп слов достаточно просто передать ей на вход строку. На выходе будет получен список объектов, в каждом из которых содержится по одному из токенов, их принадлежность к стоп-словам из словаря, начальная и исходная формы. С помощью этих объектов удобно записать в результирующую строку начальные формы токенов, которые не являются стоп-словами.

Пример применения библиотеки SpaCy к одной строке русского языка имеет следующий вид 10.

```
result = " ".join(
    [
        token.lemma_
        for token in
            self.nlp_en(self.__processing_token__(russian_str))
            if
        not (token.is_stop) and not (token.is_punct) and
            len(token.lemma_) > 1
    ]
)
```

Листинг 10: Пример применения библиотеки SpaCy для обработки одной строки русского языка

Реализация полного алгоритма, сожержащего описанные выше функции представлена в следующем листинге 11.

```
def __processing_cell__(self, cell: str) -> str:
    parts = self.__split_into_en_and_ru__(cell)
    tokens = []
    for part in parts:
        if part[0]:
            tokens += [
```

```
                    token.lemma_
                    for token in
                            self.nlp_en(self.__processing_token__(part[1]))
                    if not (token.is_stop) and not (token.is_punct)
                            and
                    len(token.lemma_) > 1
                ]
            else:
                tokens += [
                    token.lemma_
                    for token in
                            self.nlp_ru(self.__processing_token__(part[1]))
                    if not (token.is_stop) and not (token.is_punct)
                            and
                    len(token.lemma_) > 1
                ]
        return " ".join(tokens)
```

Листинг 11: Функция удаления лишних пробелов и переносов строк, токенизации, лемматизации и удаления стоп-слов по словарю

2.2.5 Удаление стоп-слов с помощью метрики tfidf

Как говорилось ранее удаление стоп-слов только по словарю не может быть исчерпывающим, поэтому можно применить метрику tfidf для расчёта значимости слов и удалять слова с малой значимостью.

Расчёт этой метрики удобно с помощью библиотеки Gensim. Для этого нужно вычислить по коллекции документов словарь, затем по словарю сформировать частотный словаь — corpus, а уже по нему вычислить tfidf метрики для слов.

Реализация соответствующей функции представлена в следующем листинге 12.

```
def
        calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictionary(s
        -> None:
    texts = []
    self.original_tokens = []
    for row in range(self.p_data.shape[0]):
        words = []
        for column in self.processing_columns:
```

```
            for word in self.p_data.loc[row, column].split(" "):
                words.append(word)
        self.original_tokens.append(words)
        texts.append(words)
    dictionary = gensim.corpora.Dictionary(texts)
    corpus = [dictionary.doc2bow(text) for text in texts]
    tfidf = gensim.models.TfidfModel(corpus)
    self.tfidf_corpus = tfidf[corpus]
    self.tfidf_dictionary = dictionary
```

Листинг 12: Функция вычисления tfidf метрики для слов документов

Однако данное вычисление не является полным, так как библиотека Gensim не добавляет в словарь слова, значение tfidf которых точно будет равняться нулю. В таком случае необходимо добавить недостающие слова. Реализация соответствующей функции представлена в следующем листинге 13.

```
def add_in_tfidf_corpus_zero_score_tokens(self) -> None:
    full_corpus = []
    for doc_idx, doc in enumerate(self.tfidf_corpus):
        original_words = self.original_tokens[doc_idx]
        term_weights = {self.tfidf_dictionary.get(term_id):
                weight for term_id, weight in doc}
        full_doc = []
        for word in original_words:
            if word in term_weights:
                weight = term_weights[word]
            else:
                weight = 0.0
            full_doc.append((word, weight))
        full_corpus.append(full_doc)
    self.tfidf_corpus = full_corpus
```

Листинг 13: Функция добавление недостающих tfidf слов

Последним шагом перед удалением стоп-слов является вычисление границы, по которой будет определяться принадлежность к стоп-словам. Сделать это можно следующим образом 14.

```
def add_in_tfidf_corpus_zero_score_tokens(self) -> None:
    full_corpus = []
    for doc_idx, doc in enumerate(self.tfidf_corpus):
        original_words = self.original_tokens[doc_idx]
```

```
        term_weights = {self.tfidf_dictionary.get(term_id):
                weight for term_id, weight in doc}
        full_doc = []
        for word in original_words:
            if word in term_weights:
                weight = term_weights[word]
            else:
                weight = 0.0
            full_doc.append((word, weight))
        full_corpus.append(full_doc)
    self.tfidf_corpus = full_corpus
```

Листинг 14: Функция вычисления tfidf границы

В данном случае к стоп-словам будут относиться слова, значение tfidf метрики которых будет относится к n минимальным процентам значений.

Теперь осталось только пройтись по датасету и удалить соответствующие стоп-слова. Реализация соответсвующей функции представлена в следующем листинге 15.

```
def del_tfidf_stop_words(self, tfidf_percent_treshold) -> None:
    self.calc_tfidf_corpus_without_zero_score_tokens_and_tfidf_dictionary
    self.add_in_tfidf_corpus_zero_score_tokens()
    self.calc_threshold_for_tfidf_stop_words(tfidf_percent_treshold)
    for row, doc in zip(range(self.p_data.shape[0]),
            self.tfidf_corpus):
        tfidf_stop_words = [word for word, tfidf_value in doc if
                tfidf_value < self.threshold_for_tfidf_stop_words]
        for column in self.processing_columns:
            words_without_tfidf_stop_words = []
            for word in self.p_data.loc[row, column].split(" "):
                if word in tfidf_stop_words:
                    continue
                words_without_tfidf_stop_words.append(word)
            self.p_data.loc[row, column] = "
                    ".join(words_without_tfidf_stop_words)
```

Листинг 15: Функция удаление вычисленных по метрике tfidf стоп-слов

Также стоит сказать, что также дополнительно стоит добавить удаление низкочастотных слов, так как это может положительно повлиять на результаты тематического моделирования.

Полный код обработчика новостного массива можно увидеть в соответствующем приложении Б.

2.3 Количественные характеристики обработанного и необработанного датасета

В рамках данной работы была выполнена обработка новостного массива с различными параметрами (имеется ввиду разные пороги для tfidf метрик, а также некоторые другие). Количественные характеристики представлены в соответствующих таблицах В.

2.4 Вычисление тематической модели

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

Листинг вебскраппера

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import os
import time
import threading

def __loading_bar_and_info__(
    start: bool, number_of_steps: int, total_steps: int,
            number_of_thread: int
) -> None:
    '''Вывод инфомрации о прогрессе выполнения программы.
    start - нужно ли вывести начальную строку;
    number_page - количество спаршенных страниц;
    total_pages - всего стнраниц, которые нужно спарсить;
    miss_count - число новостей, которые не удалось спарсить;
    whitour_whole_content - число новостей, у которых не получило
            сь полностью спарсить контент.'''
    done = int(number_of_steps / total_steps * 100) if int(
        number_of_steps / total_steps * 100
    ) < 100 or number_of_steps == total_steps else 99
    stars = int(
```

```
21          40 / 100 * done
22      ) if int(20 / 100 * done) < 20 or number_of_steps ==
                total_steps else 39
23      tires = 40 - stars
24
25      if start:
26          stars = 0
27          tires = 40
28          done = 0
29
30      print("thread{0} <".format(number_of_thread), end="")
31      for i in range(stars):
32          print("*", end="")
33
34      for i in range(tires):
35          print("-", end="")
36      print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
                total_steps))
37
38  def __getPage__(url: str, file_name: str) -> None:
39      '''Получение html файла страницы.
40      url - ссылка на страницу;
41      file_name - имя файла, в который будет сохранена страница.'''
42      r = requests.get(url=url)
43
44      with open(file_name, "w", encoding="utf-8") as file:
45          file.write(r.text)
46
47  def __parse_news__(url: str) -> str:
48      '''Получиние полного контента новости.
49      url - ссылка но новость.
50      Функция возвращает полный текст новости.'''
51      news_file_name = "news.html"
52      __getPage__(url, news_file_name)
53
54      with open(news_file_name, encoding="utf-8") as file:
55          src = file.read()
56
57      content = BeautifulSoup(src, "lxml").find("div",
                class_="main").find(
58          "div", class_="post__text"
```

```
59      ).text.strip()
60
61      return content
62
63  def __parse_page__(page_file_name: str, news_container:
        pd.DataFrame) -> None:
64      '''Парсинг информации с новостной страницы: ссылка на новость
            + короткая информация о ней.
65      page_file_name - имя файла, в который сохранён код страницы;
66      news_container - таблица, в которую заносится информация о но
            вости.
67      Функция также возвращает количество новостей, которые не удал
            ось спарсить
68      и количество новостей, полный контент которых спарсить не уда
            лось.'''
69      with open(page_file_name, encoding="utf-8") as file:
70          src = file.read()
71
72      soup = BeautifulSoup(src, "lxml")
73
74      news = soup.find("div", class_="post")
75      for i in range(10):
76          try:
77              news_day = news.find("div",
                        class_="post-meta__day").text.strip()
78          except:
79              news_day = ""
80
81          try:
82              news_month = news.find("div",
83                                     class_="post-meta__month").text.strip(
84          except:
85              news_month = ""
86
87          try:
88              news_year = news.find("div",
                        class_="post-meta__year").text.strip()
89          except:
90              news_year = ""
91
92          news_date = news_day + "." + news_month + "." + news_year
```

```
93
94          try:
95              news_name = news.find("h2",
96                                    class_="first_child").find("a").text.st
97          except:
98              news_name = ""
99
100         try:
101             news_short_content = news.find("p",
                    class_="first_child"
102                                            ).find_next_sibling("p").text.s
103         except:
104             news_short_content = ""
105
106         try:
107             link = news.find("h2",
                    class_="first_child").find("a").get("href")
108             if not link.startswith("https://"):
109                 link = 'https://www.hse.ru' + link
110         except:
111             link = ""
112
113         try:
114             if link.startswith("https://www.hse.ru/news/"):
115                 news_content = __parse_news__(link)
116         except:
117             news_content = ""
118
119         if len(
120             news_day + news_month + news_year + news_name +
                    news_short_content +
121             news_content
122         ) > 0:
123             news_container.loc[len(news_container.index)] = [
124                 link, news_date, news_name, news_short_content,
                        news_content
125             ]
126
127         news = news.find_next_sibling("div", class_="post")
```

Листинг 16: Полный код вебскраппера

ПРИЛОЖЕНИЕ Б

Листинг обработчика новостного массива

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
import os
import time
import threading

def __loading_bar_and_info__(
    start: bool, number_of_steps: int, total_steps: int,
            number_of_thread: int
) -> None:
    '''Вывод инфомрации о прогрессе выполнения программы.
    start - нужно ли вывести начальную строку;
    number_page - количество спаршенных страниц;
    total_pages - всего стнраниц, которые нужно спарсить;
    miss_count - число новостей, которые не удалось спарсить;
    whitour_whole_content - число новостей, у которых не получило
            сь полностью спарсить контент.'''
    done = int(number_of_steps / total_steps * 100) if int(
        number_of_steps / total_steps * 100
    ) < 100 or number_of_steps == total_steps else 99
    stars = int(
        40 / 100 * done
    ) if int(20 / 100 * done) < 20 or number_of_steps ==
            total_steps else 39
    tires = 40 - stars

    if start:
        stars = 0
        tires = 40
        done = 0

    print("thread{0} <".format(number_of_thread), end="")
    for i in range(stars):
        print("*", end="")

    for i in range(tires):
        print("-", end="")
    print("> {0}% ||| {1} / {2}".format(done, number_of_steps,
```

```
            total_steps))

def __getPage__(url: str, file_name: str) -> None:
    '''Получение html файла страницы.
    url - ссылка на страницу;
    file_name - имя файла, в который будет сохранена страница.'''
    r = requests.get(url=url)

    with open(file_name, "w", encoding="utf-8") as file:
        file.write(r.text)

def __parse_news__(url: str) -> str:
    '''Получиние полного контента новости.
    url - ссылка но новость.
    Функция возвращает полный текст новости.'''
    news_file_name = "news.html"
    __getPage__(url, news_file_name)

    with open(news_file_name, encoding="utf-8") as file:
        src = file.read()

    content = BeautifulSoup(src, "lxml").find("div",
            class_="main").find(
        "div", class_="post__text"
    ).text.strip()

    return content

def __parse_page__(page_file_name: str, news_container:
        pd.DataFrame) -> None:
    '''Парсинг информации с новостной страницы: ссылка на новость
            + короткая информация о ней.
    page_file_name - имя файла, в который сохранён код страницы;
    news_container - таблица, в которую заносится информация о но
            вости.
    Функция также возвращает количество новостей, которые не удал
            ось спарсить
    и количество новостей, полный контент которых спарсить не уда
            лось.'''
    with open(page_file_name, encoding="utf-8") as file:
        src = file.read()
```

```
71
72      soup = BeautifulSoup(src, "lxml")
73
74      news = soup.find("div", class_="post")
75      for i in range(10):
76          try:
77              news_day = news.find("div",
                        class_="post-meta__day").text.strip()
78          except:
79              news_day = ""
80
81          try:
82              news_month = news.find("div",
83                                     class_="post-meta__month").text.strip(
84          except:
85              news_month = ""
86
87          try:
88              news_year = news.find("div",
                        class_="post-meta__year").text.strip()
89          except:
90              news_year = ""
91
92          news_date = news_day + "." + news_month + "." + news_year
93
94          try:
95              news_name = news.find("h2",
96                                    class_="first_child").find("a").text.st
97          except:
98              news_name = ""
99
100         try:
101             news_short_content = news.find("p",
                        class_="first_child"
102                                            ).find_next_sibling("p").text.s
103         except:
104             news_short_content = ""
105
106         try:
107             link = news.find("h2",
                        class_="first_child").find("a").get("href")
```

```
            if not link.startswith("https://"):
                link = 'https://www.hse.ru' + link
        except:
            link = ""

        try:
            if link.startswith("https://www.hse.ru/news/"):
                news_content = __parse_news__(link)
        except:
            news_content = ""

        if len(
            news_day + news_month + news_year + news_name +
                news_short_content +
            news_content
        ) > 0:
            news_container.loc[len(news_container.index)] = [
                link, news_date, news_name, news_short_content,
                    news_content
            ]

        news = news.find_next_sibling("div", class_="post")
```

Листинг 17: Полный код подготовки новостного массива

ПРИЛОЖЕНИЕ В

Количественные характеристики подготовленного и неподготовленного новостного массива

Характеристика	Неподгот.	Стоп-слова	+Низкочаст.	TF-IDF 1%	TF-IDF 2%	TF-IDF 3%
Кол. док.	17340	17340	17340	17340	17340	17340
Кол. токенов	1213111	6545045	-	6479545	6414045	6348544

Продолжение следует...

Продолжение таблицы

Характеристика	**Неподгот.**	**Стоп-слова**	**+Низкочаст.**	**TF-IDF 1%**	**TF-IDF 2%**	**TF-IDF 3%**
Кол. уник. ток.	278724	148677	-	148677	148677	148677
Мин. кол. ток. в док.	6	4	-	4	4	4
Модальное кол. ток. в док.	47	31	-	31	31	30
Среднее кол. ток. в док.	695	375	-	371	367	364
Медианное кол. ток. в док.	-	313	-	312	310	309
Макс. кол. ток. в док.	6514	3151	-	2903	2825	2766
Мин. кол. уник. ток. в док.	6	4	-	4	4	4
Мод. кол. уник. ток. в док.	39	27	-	27	27	30
Сред. кол. уник. ток. в док.	346	214	-	211	208	205
Мед. кол. уник. ток. в док.	-	186	-	185	183	182

Продолжение следует...

Продолжение таблицы

Характеристика	**Неподгот.**	**Стоп-слова**	**+Низкочаст.**	**TF-IDF 1%**	**TF-IDF 2%**	**TF-IDF 3%**
Макс. кол. уник. ток. в док.	2287	1353	-	1299	1262	1214

Характеристика	**TF-IDF 4%**	**TF-IDF 5%**	**TF-IDF 6%.**	**TF-IDF 7%**	**TF-IDF 8%**	**TF-IDF 9%**
Кол. док.	17340	17340	17340	17340	17340	17340
Кол. токенов	6283046	6217544	6152044	6086544	6021044	5955543
Кол. уник. ток.	148677	148677	148677	148677	148677	148677
Мин. кол. ток. в док.	4	4	4	4	4	4
Модальное кол. ток. в док.	30	30	30	30	29	29
Среднее кол. ток. в док.	360	356	352	349	345	341
Медианное кол. ток. в док.	307	306	305	303	301	299

Продолжение следует...

Продолжение таблицы

Характеристика	**TF-IDF 4%**	**TF-IDF 5%**	**TF-IDF 6%**	**TF-IDF 7%**	**TF-IDF 8%**	**TF-IDF 9%**
Макс. кол. ток. в док.	2713	2662	2595	2545	2501	2424
Мин. кол. уник. ток. в док.	4	4	4	4	4	4
Мод. кол. уник. ток. в док.	27	29	29	28	28	28
Сред. кол. уник. ток. в док.	201	198	195	192	189	186
Мед. кол. уник. ток. в док.	181	179	177	176	174	172
Макс. кол. уник. ток. в док.	1164	1122	1085	1047	1018	986

Характеристика	TF-IDF 10%	TF-IDF 10% + Низк.
Кол. док.	17340	17340
Кол. токенов	5890042	-
Кол. уник. ток.	148677	-
Мин. кол. ток. в док.	4	-
Модальное кол. ток. в док.	30	-
Среднее кол. ток. в док.	337	-
Медианное кол. ток. в док.	297	-
Макс. кол. ток. в док.	2391	-
Мин. кол. уник. ток. в док.	4	-
Мод. кол. уник. ток. в док.	28	-
Сред. кол. уник. ток. в док.	182	-

Продолжение следует...

Продолжение таблицы

Характеристика	**TF-IDF 10%**	**TF-IDF 10% + Низк.**
Мед. кол. уник. ток. в док.	170	-
Макс. кол. уник. ток. в док.	946	-