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 Б.

2.2 Подготовка новостного массива

2.2.1 Очистка от лишних пробелов и переноса строк

Воизбежание некорректной токенизации и просто ради удобства просмотра коллекции документов необходимо провести удаление лишних пробелов и переносов строк. Сделать это можно с помощью стандартных средств языка python. Код соответствующего алгоритма представлен в следующем листинге 8.

```
1  def __remove_extra_spaces_and_line_breaks__(self, text: str) ->
       str:
2      processed = ""
3      # если ячейка текста пуста возвращаем пустую строку
4      if type(text) != str or len(text) == 0:
5          return ""
6      flag = True
7      for symb in text:
8          # при встрече пробела или разрыва строки пропускаем все
9          # последующие пробелы или разрывы строки
10         if flag and (symb == " " or symb == "\n"):
11             processed += " "
12             flag = False
13         # пока не встретим отличающийся символ
14         if symb != " " and symb != "\n":
15             flag = True
16         if flag:
17             processed += symb
```

```
18      return processed.strip()
```

Листинг 7: Функция удаления лишних пробелов и переносов строк

2.2.2 Разделение строки фрагменты с русским и английским

Библиотека SpaCy является нейросетевой. Она предоставляет различные предобученные модели, как для русского языка, так и для английского, но не для мультиязычного. Из-за этого возникает проблема, так как в выбранном новостном массиве присутствует текст на английском, а модель, предобученная для русского обработать его не сможет.

Решить эту проблему можно, если обрабатывать русский и английский текст по отдельности. Для этого нужно разбивать строку на подстроки только с русским или английским текстом. Реализовать это можно с помощью стандартных средств языка python.

Алгоритм действия будет следующим:

1. Ищем первую букву в исходной последовательности;
2. Определяем к какому алфавиту принадлежит буква;
3. Далее записываем в подстроку символы, пока не встретим букву из другого алфавита;
4. Если была встречена буква из другого алфавита, то сохраняем кортеж вида (тип алфавита, подстрока) в соответствующий массив, очищаем входную последовательность, изменяем тип алфавита и переходим к пункту 3. Так выполняем, пока исходная строка не пройдена полностью.

Реализация соответствующих функций представлена в следующем листинге.

```
1  def __first_is_en__(self, cell: str) -> bool:
2          index_first_en = re.search(r"[a-zA-Z]", cell)
3          index_first_ru = re.search(r"[а-яА-Я]", cell)
4          return True if index_first_en and (not(index_first_ru)
                   or index_first_en.start() <
                   index_first_ru.start()) else False
5  def __split_into_en_and_ru__(self, cell: str) -> list[(bool,
       str)]:
6      parts = []
7      is_en = self.__first_is_en__(cell)
8      part = ""
9      for symb in cell:
```

```
10          if is_en == (symb in string.ascii_letters) or not
                (symb.isalpha()):
11              part += symb
12          else:
13              parts.append((is_en, part))
14              part = symb
15              is_en = not (is_en)
16      if part:
17          parts.append((is_en, part))
18      return parts
```

Листинг 8: Функция удаления лишних пробелов и переносов строк

2.2.3 Обработка временных меток и двоеточий

Важно также обработать все токены, содержащие двоеточия. Так как для BigARTM тематической модели двоеточие является спецсимволом, то присутствие его в тексте может приветсти к ошибкам в вычислении тематической модели.

Двоеточие может указывать как на время, так и использоваться в других случаях, значение которых угадать будет трудно. Тогда все случаи употребления символа двоеточия во временных метках будем заменять на строку time, а в остальных случая будем просто данный символ удалять. Таким образом, мы обработаем временные метки и удалим лишние двоеточия.

Реализация соответствующей функции представлена в следующем листинге.

2.2.4 Токенизация, лемматизация, очистка от стоп-слов

После того как исходная последовательность разбита на русские и английские подстроки её можно обработать. Обработка будет происходить с помощью библиотеки SpaCy. Для этого ей достаточно просто передать последовательность, а она уже внутри себя токенизирует последовательность, приведёт токены к начальной форме, посредством лемматизации, а также по заданному в ней словарю определит принадлежность каждого из слов к стоп-словам. На выходе получится набор объектов, в каждом из которых содержится по одному из токентов, его характеристи и начальная форма. Набор будет иметь тот же порядок, что и входная последовательность.

Реализация

ЗАКЛЮЧЕНИЕ

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

ПРИЛОЖЕНИЕ А

Нумеруемые объекты в приложении

ПРИЛОЖЕНИЕ Б

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(
        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

```

```
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      '''Получчиние 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"
```

```
                                                ).find_next_sibling("p").text.s
        except:
            news_short_content = ""

        try:
            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 9: Полный код вебскраппера