МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | **Автоматики и вычислительной техники** |
| Кафедра | **Информатики** |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Оценка комиссии: |  | | | Рейтинг: |  |
| Подписи членов комиссии: | | | | | |
|  | |  | Вишневская Е.А. | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | |  |  | | |
| (подпись) | |  | (фамилия, имя, отчество) | | |
|  | | | | | |
| (дата) | | | | | |
|  | |  |  | | |

**КУРСОВАЯ РАБОТА**

|  |  |
| --- | --- |
| по дисциплине | Средства интеллектуального анализа данных и машинное |
| обучение | |

|  |  |
| --- | --- |
| на тему | Создание электронной библиотеки научных статей с применением |
| нейронной сети. | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| «К ЗАЩИТЕ» |  | ВЫПОЛНИЛ: |  |
|  |  | Студент группы | **АА-22-08** |
|  |  |  | (номер группы) |
| к.ф.м.н, доцент кафедры информатики,  Вишневская Е.А. |  | Сафуанов Артур Ришатович | |
| (должность, ученая степень; фамилия, и.о.) |  | (фамилия, имя, отчество) | |
|  |  |  | |
| (подпись) |  | (подпись) | |
|  |  |  | |
| (дата) |  | (дата) | |

|  |  |  |  |
| --- | --- | --- | --- |
|  | Москва, 20 | 25 |  |

МИНОБРНАУКИ РОССИИ

РГУ нефти и газа (НИУ) имени И.М. Губкина

|  |  |
| --- | --- |
| Факультет | **Автоматики и вычислительной техники** |
| Кафедра | **Информатики** |

**ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| по дисциплине | Средства интеллектуального анализа данных и |
| машинное обучение | |

|  |  |
| --- | --- |
| на тему | Создание электронной библиотеки научных статей с применением |
| нейронной сети. | |
|  | |

|  |  |  |  |
| --- | --- | --- | --- |
| ДАНО студенту | **Сафуанову Артуру Ришатовичу** | группы | **АА-22-08** |
|  | (фамилия, имя, отчество в дательном падеже) |  | (номер группы) |

**Содержание работы:**

|  |  |
| --- | --- |
|  | Введение |
|  | Теоретическая часть |
|  | Практическая часть |
|  | Заключение |
|  | Список используемых источников |
|  | Приложения |

**Исходные данные для выполнения работы:**

|  |  |
| --- | --- |
|  | Данные с сайта elibrary.ru |

**Рекомендуемая литература:**

|  |  |
| --- | --- |
|  | Бёрд С., Кляйн Э., Лопер Э. Обработка естественного языка на Python. Анализ текста с помощью Natural Language Toolkit / Пер. с англ. – СПб.: Символ-Плюс, 2011. – 504 с. |
|  | Шолле Ф. Глубокое обучение на Python / Пер. с англ. – М.: ДМК Пресс, 2022. – 384 с. |

**Графическая часть:**

|  |  |
| --- | --- |
|  |  |

**Требования к представлению результатов:**

|  |  |
| --- | --- |
| ✓ | Электронная версия |
|  | Бумажный вариант и электронный образ документа |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Руководитель: | к.ф.м.н. |  | Доцент |  |  |  | Вишневская Е.А. |
|  | (уч.степень) |  | (должность) |  | (подпись) |  | (фамилия, имя, отчество) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Задание принял к исполнению: | студент |  |  |  | Сафуанов А.Р. |
|  | |  | (подпись) |  | (фамилия, имя, отчество) |

**СОДЕРЖАНИЕ**

[ВВЕДЕНИЕ 5](#_Toc198480624)

[ТЕОРЕТИЧЕСКАЯ ЧАСТЬ 8](#_Toc198480625)

[ПРАКТИЧЕСКАЯ ЧАСТЬ 12](#_Toc198480627)

[ЗАКЛЮЧЕНИЕ 19](#_Toc198480628)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 21](#_Toc198480629)

[ПРИЛОЖЕНИЯ 22](#_Toc198480630)

[Приложение 1 22](#_Toc198480631)

[Приложение 2 31](#_Toc198480632)

# Введение

**Актуальность исследования**

В условиях стремительного роста объема научных публикаций традиционные методы организации и классификации статей становятся неэффективными. Ежегодно в базах данных, таких как elibrary.ru, Scopus и Web of Science, появляются миллионы новых работ, что делает ручную обработку информации практически невозможной. Современные исследователи сталкиваются с рядом проблем:

1. **Трудоемкость** – традиционные методы требуют значительных временных затрат и подвержены человеческим ошибкам.
2. **Динамичность научных направлений** – появление новых областей исследований требует гибких систем классификации.
3. **Неоднородность форматов** – статьи отличаются структурой, стилем изложения и наличием технических элементов (формулы, графики), что усложняет автоматическую обработку.

Применение **искусственного интеллекта** и **нейронных сетей** для создания электронных библиотек позволяет решить эти проблемы. Такие системы способны:

* Автоматически определять тематику статей с высокой точностью.
* Адаптироваться к новым научным направлениям без перестройки алгоритмов.
* Обеспечивать быстрый поиск и фильтрацию материалов.

**Цель работы**

Целью курсовой работы является разработка прототипа электронной библиотеки научных статей с автоматической классификацией материалов с помощью нейронной сети.

Для достижения этой цели поставлены следующие задачи:

1. **Анализ существующих решений**: изучение современных методов обработки естественного языка (NLP) и их применимости для классификации научных текстов.
2. **Сбор и предобработка данных**: загрузка статей из открытых источников (например, elibrary.ru) и их очистка от служебной информации (авторы, библиография, технические элементы).
3. **Выбор и настройка модели**: тестирование нейросетевых архитектур (в частности, mDeBERTa-v3 от Hugging Face) для задач zero-shot классификации.
4. **Интеграция с базой данных**: сохранение результатов классификации в PostgreSQL для дальнейшего использования.
5. **Оценка эффективности**: проверка точности модели на реальных данных и сравнение с альтернативными методами.

Объект исследования: процесс автоматической классификации научных статей.

Предмет исследования: применение нейронных сетей для категоризации текстов в электронных библиотеках.

**Структура работы**

Курсовая работа состоит из следующих разделов:

1. **Введение**

Обоснование актуальности темы, формулировка цели и задач, описание структуры работы.

1. **Теоретическая часть**

Обзор современных методов NLP для классификации текстов. Архитектура трансформеров и BERT. Zero-shot классификация.

1. **Практическая часть**

Описание процесса сбора данных (парсинг PDF, очистка текстов).Реализация пайплайна: от предобработки до интеграции с БД. Анализ результатов: точность, скорость работы, ограничения модели.

1. **Заключение**

Итоги работы, выводы о применимости выбранного подхода, рекомендации по улучшению системы.

1. **Список использованных источников**

Научные статьи, документация библиотек, учебные пособия.

1. **Приложения**

Листинги кода, примеры обработанных статей, скриншоты интерфейса (если есть).

# Теоретическая часть

Современные достижения в области искусственного интеллекта и обработки естественного языка (NLP) открывают новые возможности для автоматизации работы с научными текстами. В этой части работы рассматриваются ключевые теоретические аспекты, лежащие в основе разработки электронной библиотеки с применением нейронной сети.

**1. Основы обработки естественного языка (NLP)**

Обработка естественного языка (NLP) — это область искусственного интеллекта, направленная на взаимодействие компьютеров с человеческим языком. Для классификации текстов используются следующие подходы:

* **Правила и ключевые слова**: устаревший метод, основанный на поиске шаблонов.
* **Машинное обучение**: алгоритмы вроде SVM или Random Forest, обучающиеся на TF-IDF или Bag-of-Words.
* **Глубокое обучение**: нейронные сети (RNN, CNN, трансформеры), способные улавливать контекст.

Математическая основа:

TF-IDF (Term Frequency-Inverse Document Frequency):

Где N — общее число документов, DF(t) — число документов с термином t.

**2. Архитектура трансформеров**

Трансформеры — нейросетевые архитектуры, использующие механизм self-attention вместо рекуррентных слоев. Основные компоненты:

* Self-Attention**:** вычисляет вес каждой пары токенов в последовательности.

Где Q (запрос), K (ключ), V (значение) — матрицы, полученные из входных данных.

* Multi-Head Attention: объединяет несколько механизмов внимания для улавливания разных аспектов контекста.

Позиционные кодировки:

Где pos — позиция токена, i — размерность.

**3. Модель BERT и её развитие**

BERT (Bidirectional Encoder Representations from Transformers) — модель, обучающаяся на задачах маскирования токенов (MLM) и предсказания следующего предложения (NSP).

DeBERTa (Decoding-enhanced BERT with Disentangled Attention) улучшает BERT за счет:

* Disentangled Attention: разделение внимания на содержание и позицию токенов.

Где Qc, Kc — векторы содержания, Qp, Kp — векторы позиций.

* Enhanced Mask Decoder: улучшенное декодирование маскированных токенов.

mDeBERTa-v3 — мультиязычная версия DeBERTa, оптимизированная для zero-shot классификации.

**4. Zero-shot классификация**

Zero-shot классификация позволяет определять категории текста без предварительного обучения на размеченных данных.

**Принцип работы**:

1. **Гипотезы**: для каждого класса генерируется шаблон, например:

*"Этот текст о {класс}."*

1. **Сравнение с текстом**: модель оценивает, насколько текст соответствует гипотезам.

Функция потерь:

Для обучения используется Cross-Entropy Loss:

Где yi ​ — истинная метка, pi ​ — предсказанная вероятность.

**5. Метрики оценки качества**

Для оценки классификации используются:

Точность (Accuracy):

F1-мера:

где:

**6.** **Предобработка текстов**

Для работы с научными статьями применяются:

* **Удаление шума**: регулярные выражения для удаления различных символов и слов на английском языке.
* **Токенизация**: разбиение текста на слова.
* **Стемминг/Лемматизация**: приведение слов к базовой форме.

**7. Интеграция с базой данных**

Для хранения классифицированных статей используется PostgreSQL.

# Практическая часть

1. **Сбор данных**

На первом этапе работы был проведен сбор научных статей для формирования датасета. Основным источником выступила платформа **elibrary.ru** — крупнейшая российская электронная библиотека научных публикаций. Для обеспечения репрезентативности данных статьи были отобраны по пяти ключевым тематикам: *образование, искусство, спорт, химия* и *космос*. Каждая категория включала **20 статей**, что в сумме составило **100 документов**.

Процесс загрузки осуществлялся вручную через интерфейс сайта:

* Поиск статей выполнялся по ключевым словам, соответствующим выбранным темам (например, "спорт", "химия").
* PDF-файлы сохранялись в структурированные папки в корне проекта. Такая организация данных упростила дальнейшую обработку и проверку корректности классификации.

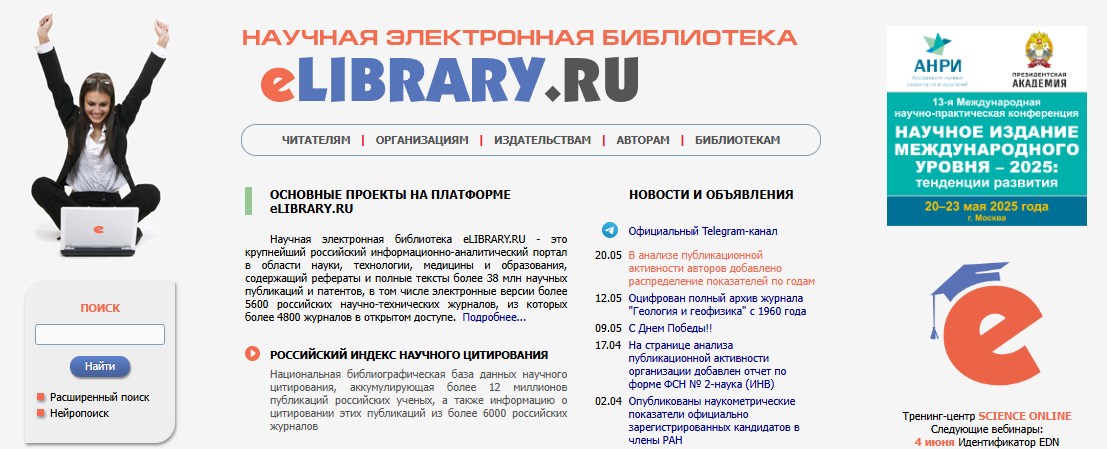


Рисунок 2.1. главная страница сайта elibrary.ru

1. **Предобработка текстов**

Поскольку научные статьи содержат значительное количество служебной информации, была реализована многоэтапная очистка текста. Основные шаги включали:

* Извлечение текста из PDF:  
  С использованием библиотеки PyPDF2 текст извлекался постранично, после чего объединялся в единую строку.
* Удаление технических элементов:  
  Для этого применялись регулярные выражения, которые фильтровали:

1. Номера ISSN, DOI, библиографические ссылки (например, шаблон для удаления сносок).
2. Информацию об авторах и учреждениях (например, паттерны для поиска названий университетов).
3. Разделы *Abstract*, *Keywords* и *References*, которые не несут смысловой нагрузки для классификации.

* Очистка от специальных символов:  
  Удалялись email-адреса, URL-ссылки, а также нестандартные символы (например, \xa0).

В результате предобработки тексты статей были приведены к единому формату, что повысило качество последующего анализа.

Тем самым, удалось создать готовый датасет для предсказания моделью, которая будет использоваться далее.

**3. Загрузка и тестирование модели**

Для классификации статей была выбрана модель mDeBERTa-v3 от Hugging Face — мультиязычная нейросетевая архитектура, оптимизированная для задач zero-shot классификации.

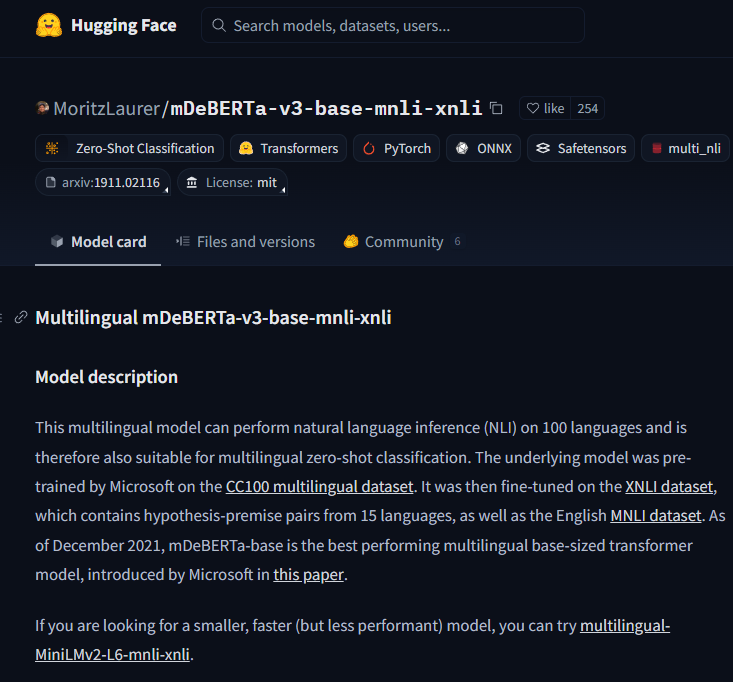


Рисунок 2.2. описание модели на HuggingFace

Её ключевые преимущества:

* Способность работать с редкими тематиками без предварительного обучения.
* Поддержка русского языка и специализированной терминологии.

Процесс тестирования:

* Модель загружалась через transformers.pipeline с указанием шаблона гипотез
* На начальном этапе модель проверялась на нескольких статьях для оценки её "понимания" контекста.

При предсказании класса использовалась взвешенная метрика, которая суммировала уверенность модели на каждой части статьи, а в конце возвращался класс с самым большим абсолютным значением.

На процессе выполнения этого пункта возникла проблема со скоростью предсказания класса, на некоторых документах код мог выполняться до 7 минут, поэтому было решено перейти на платформу Kaggle, где предоставляются бесплатные вычислительные мощности:



Рисунок 2.3. скорость обработки статьи на своём процессоре



Рисунок 2.4. скорость обработки статьи на видеокартах Kaggle

Таким образом, общая обработка всех 100 научных статей заняла 29 минут, что значительно быстрее, чем на CPU.

**4. Формирование CSV-файла с предсказаниями**

Результаты классификации всех статей были сохранены в CSV-файл predictions.csv, где:

* Каждая строка соответствовала одной статье.
* Столбцы отражали пять тематических категорий.



Рисунок 2.5. файл preds.csv

Файл использовался для последующего анализа точности модели и визуализации результатов.

**5. Оценка метрик качества**

Для проверки эффективности модели применялись стандартные метрики классификации:

* **Accuracy (Общая точность)**: доля верно предсказанных категорий относительно общего числа статей. Составила **91%**, что подтвердило высокую надежность модели.
* **Матрица ошибок (Confusion Matrix)**: визуализировала распределение ошибок по категориям. Наибольшая путаница наблюдалась между статьями об *искусстве* и *образовании*, что связано с пересечением терминологии в гуманитарных науках.
* **F1-мера**: среднее гармоническое точности и полноты. Значение **0.91** показало сбалансированность модели.

Результаты подтвердили, что zero-shot подход эффективен для задач автоматической категоризации научных текстов.

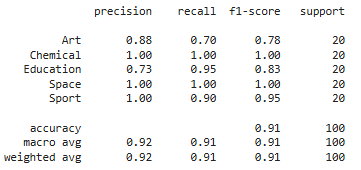


Рисунок 2.6. отчёт классификации по всем темам

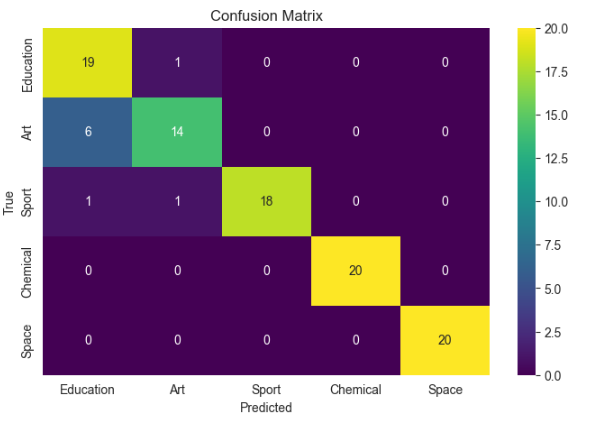


Рисунок 2.7. матрица ошибок

**6. Интеграция с PostgreSQL**

Для хранения классифицированных статей была разработана база данных в PostgreSQL.

Структура БД:

* Таблица articles содержала поля:
  + url — ссылка на статью (уникальный идентификатор).
  + Булевы флаги для каждой категории (education, art, и т.д.).

Реализация:

* Соединение с БД устанавливалось через библиотеку psycopg2.
* Для каждой статьи генерировался SQL-запрос, который добавлял её в таблицу или обновлял данные при повторе.



Рисунок 2.8. процесс записи научной статьи в базу данных

**7. Наполнение базы данных**

На финальном этапе в БД были загружены выборочно 5 научных статей, по 1 на каждую тематику, чтобы убедиться в том, что модель правильно работает, а также для дальнейшего пользования. В БД по ссылке можно перейти на интересующую научную статью. Процесс включал:

* Последовательную обработку каждого PDF-файла.
* Классификацию текста моделью.
* Запись результата в соответствующую строку таблицы.

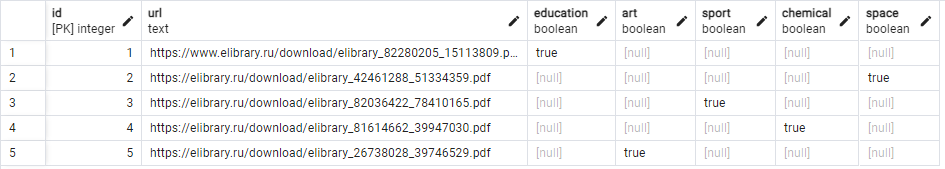


Рисунок 2.9. заполненная база данных

# Заключение

В ходе выполнения данной курсовой работы была разработана и реализована система автоматической классификации научных статей с использованием нейронной сети и интеграцией с базой данных PostgreSQL. Проект решает актуальную проблему организации больших массивов научных публикаций, обеспечивая их быструю и точную категоризацию.

В процессе работы были полностью выполнены следующие задачи:

1. **Анализ существующих решений**: изучение современных методов обработки естественного языка (NLP) и их применимости для классификации научных текстов.
2. **Сбор и предобработка данных**: загрузка статей из открытых источников (например, elibrary.ru) и их очистка от служебной информации (авторы, библиография, технические элементы).
3. **Выбор и настройка модели**: тестирование нейросетевых архитектур (в частности, mDeBERTa-v3 от Hugging Face) для задач zero-shot классификации.
4. **Интеграция с базой данных**: сохранение результатов классификации в PostgreSQL для дальнейшего использования.
5. **Оценка эффективности**: проверка точности модели на реальных данных и сравнение с альтернативными методами.

Разработанная система обладает рядом преимуществ, которые делают её полезной для реального применения:

* **Экономия времени**: Автоматизация классификации сокращает затраты на ручную обработку статей.
* **Масштабируемость**: Система легко адаптируется для новых тематик и языков.
* **Гибкость**: Интеграция с базой данных позволяет использовать решение в различных сценариях (университетские библиотеки, исследовательские платформы).

Кроме того, проект демонстрирует потенциал применения современных NLP-технологий в академической сфере, открывая возможности для дальнейших исследований и улучшений.

Данный проект подтвердил возможность эффективного применения нейронных сетей для автоматической классификации научных статей. Реализованная система не только соответствует современным требованиям к точности и скорости обработки, но и предлагает гибкие возможности для интеграции с другими инструментами.

Перспективы дальнейшего развития:

* Расширение набора категорий для классификации.
* Внедрение механизмов переобучения модели на новых данных.
* Разработка дополнительных функций.

Таким образом, работа вносит вклад в область автоматизации обработки научных текстов и может служить основой для более масштабных проектов на будущее в университете.

# Список используемых источников

1. Гудфеллоу Я., Бенджио И., Курвилль А. *Глубокое обучение*. — М.: ДМК Пресс, 2022. — 652 c.
2. Бёрд С., Кляйн Э., Лопер Э. Обработка естественного языка на Python. Анализ текста с помощью Natural Language Toolkit / Пер. с англ. – СПб.: Символ-Плюс, 2011. – 504 с.
3. Шолле Ф. Глубокое обучение на Python / Пер. с англ. – М.: ДМК Пресс, 2022. – 384 с.
4. Рашка, С. Python и машинное обучение: машинное и глубокое обучение с использованием Python, Scikit-Learn и TensorFlow : перевод с английского / С. Рашка, В. Мирджалили; редактор С. Н. Тригуб. — 2-е изд. — Санкт-Петербург: Диалектика, 2019. — 656 с.

# Приложения

## **Приложение 1**

**Методы первой необходимости**

import os

os**.**environ['HF\_HOME'] **=** 'D:\\HuggingFace'

os**.**environ['TRANSFORMERS\_CACHE'] **=** os**.**environ['HF\_HOME']

os**.**environ['HUGGINGFACE\_HUB\_CACHE'] **=** os**.**environ['HF\_HOME']

from warnings import filterwarnings

filterwarnings('ignore')

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import torch

from transformers import pipeline

import re

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

sns.set\_style('darkgrid')

device = 'cuda' if torch.cuda.is\_available() else 'cpu'

**Предобработка текстов в pdf и создание датасета**

import PyPDF2

import pathlib

clean\_data = pd.DataFrame({"Education": [], "Art": [], "Space": [], "Sport": [], "Chemical": []})

def process\_info(file):

with open(file, "rb") as f:

reader = PyPDF2.PdfReader(f)

text = ''.join([page.extract\_text() for page in reader.pages])

*# Служебная информация*

text = re.sub(r'ISSN\s+\d{4}-\d{3,4}[^\n]\*', '', text)

text = re.sub(r'\d{4};\d{2}\(\d+\):\d+–\d+', '', text)

*# Авторы*

text = re.sub(r'[А-ЯЁ][а-яё]+\s+[А-ЯЁ][\.\s]+\s\*[А-ЯЁ][\.\s]\*', '', text)

text = re.sub(r'[А-ЯЁ][а-яё]+\s+[А-ЯЁ][а-яё]+\s+[А-ЯЁ][\.\s]+\s\*[А-ЯЁ][\.\s]\*', '', text)

text = re.sub(r'\d+[\s\w\.,–-]+(университет|институт|академия|центр)[^\n]\*', '', text)

*# Сноски в квадратных скобках*

text = re.sub(r'\[\d+\]', '', text) *# [1], [2]*

text = re.sub(r'\[\d+[,-]\d+\]', '', text) *# [1-3], [4,5]*

text = re.sub(r'\[[A-Za-z]+\d\*\]', '', text) *# [A1], [B]*

*# email*

text = re.sub(r'\S+@\S+', '', text)

*# английские разделы*

text = re.sub(r'Abstract[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

text = re.sub(r'Keywords[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

text = re.sub(r'For citation[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

*# ссылки*

text **=** re**.**sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', text)

text **=** re**.**sub(r'DOI:\s\*\S+', '', text)

*# библиография*

text **=** re**.**sub(r'Список\s+источников[\s\S]\*', '', text)

text **=** re**.**sub(r'References[\s\S]\*', '', text)

*# спец.символы*

text **=** text**.**replace('\xa0', ' ')**.**replace('•', '')

text **=** re**.**sub(r'-\s+', '', text)

text **=** re**.**sub(r'\s+', ' ', text)**.**strip()

text **=** re**.**sub(r'•\s\*\n', '', text)

*# оставшиеся английские фрагменты*

text **=** re**.**sub(r'(?:[A-Za-z-]+\s){3,}[A-Za-z-]\*', '', text)

return text

**Тексты образования**

books **=** pathlib**.**Path("D:/ПРОГА/Проектики/Github/Electronic\_library/dataset/Образование")

files\_to\_process **=** []

for book in books**.**iterdir():

files\_to\_process**.**append(str(book))

clean\_files **=** []

for file in files\_to\_process:

clean\_files**.**append(process\_info(file))

clean\_data['Education'] **=** clean\_files

**Тексты спорта**

books **=** pathlib**.**Path("D:/ПРОГА/Проектики/Github/Electronic\_library/dataset/Спорт")

files\_to\_process **=** []

for book in books.iterdir():

files\_to\_process.append(str(book))

clean\_files = []

for file in files\_to\_process:

clean\_files**.**append(process\_info(file))

clean\_data['Sport'] **=** clean\_files

**Тексты космоса**

books **=** pathlib**.**Path("D:/ПРОГА/Проектики/Github/Electronic\_library/dataset/Космос")

files\_to\_process **=** []

for book in books.iterdir():

files\_to\_process.append(str(book))

clean\_files = []

for file in files\_to\_process:

clean\_files.append(process\_info(file))

clean\_data['Space'] **=** clean\_files

**Тексты химии**

books **=** pathlib**.**Path("D:/ПРОГА/Проектики/Github/Electronic\_library/dataset/Химия")

files\_to\_process **=** []

for book in books.iterdir():

files\_to\_process.append(str(book))

clean\_files = []

for file in files\_to\_process:

clean\_files**.**append(process\_info(file))

clean\_data['Chemical'] **=** clean\_files

**Тексты искусствоведения**

books **=** pathlib**.**Path("D:/ПРОГА/Проектики/Github/Electronic\_library/dataset/Искусствоведение")

files\_to\_process **=** []

for book in books.iterdir():

files\_to\_process.append(str(book))

clean\_files = []

for file in files\_to\_process:

clean\_files**.**append(process\_info(file))

clean\_data['Art'] **=** clean\_files

clean\_data**.**shape

**Создание датасета**

clean\_data**.**to\_csv('clean\_data.csv', sep**=**',', index**=**False, encoding**=**'utf-8-sig')

**Построение модели**

**Загрузка данных**

data **=** pd**.**read\_csv('clean\_data.csv')

data**.**head()

**Загрузка модели с Hugging Face**

pipe **=** pipeline("zero-shot-classification", model**=**"MoritzLaurer/mDeBERTa-v3-base-mnli-xnli", hypothesis\_template**=**"Определи тему научной статьи '{}'", disable\_hf\_device\_logging**=**True)

**Разделение текста на части**

from nltk import sent\_tokenize

def chunk\_text(text, window\_size**=**3, overlap**=**1):

sentences **=** sent\_tokenize(text)

chunks **=** []

for i in range(0, len(sentences), window\_size-overlap):

chunks.append(" ".join(sentences[i:i+window\_size]))

return chunks

chunks **=** chunk\_text(data['Education'][0])

**Проверка модели на частях**

classes **=** ['Education', 'Art', 'Space', 'Sport', 'Chemical']

def get\_pred(query):

answer **=** pipe(query, classes, multi\_label**=**False)

return [answer['labels'], answer['scores']]

print(get\_pred(chunks[0]))

print(get\_pred(chunks[1]))

print(get\_pred(chunks[2]))

print(get\_pred(chunks[3]))

from collections import defaultdict

def aggregate(document):

*# Делим документ*

chunks **=** chunk\_text(document)

*# Предсказываем каждую часть*

preds **=** []

for chun in chunks:

preds.append(get\_pred(chun))

*# Считаем взвешенную метрику*

scores = defaultdict(float)

for pred in preds:

for label, score in zip(pred[0], pred[1]):

scores[label] += score

return max(scores, key=scores.get)

*# Засекаем начальное время*

start\_time **=** time**.**time()

print(aggregate(data['Education'][0]))

*# Засекаем конечное время и вычисляем разницу*

end\_time **=** time**.**time()

execution\_time **=** end\_time **-** start\_time

print(f"Код выполнялся {execution\_time:.4f} секунд")

**Предсказание класса на научных статьях**

predictions **=** pd**.**DataFrame()

**Предсказания проделываются на Kaggle, так как там значительно быстрее на GPU**

import time

*# Засекаем начальное время*

start\_time **=** time**.**time()

education **=** []

art **=** []

sport **=** []

chemical **=** []

space **=** []

for i in range(20):

education.append(aggregate(data['Education'][i]))

art.append(aggregate(data['Art'][i]))

sport**.**append(aggregate(data['Sport'][i]))

chemical**.**append(aggregate(data['Chemical'][i]))

space**.**append(aggregate(data['Space'][i]))

print(f'{i} iteration')

predictions['Education'] **=** education

predictions['Art'] **=** art

predictions['Sport'] **=** sport

predictions['Chemical'] **=** chemical

predictions['Space'] **=** space

*# Засекаем конечное время и вычисляем разницу*

end\_time **=** time**.**time()

execution\_time **=** end\_time **-** start\_time

print(f"Код выполнялся {execution\_time:.4f} секунд")

**Загрузка предсказаний с Kaggle**

preds **=** pd**.**read\_csv('preds.csv')**.**drop('Unnamed: 0', axis**=**1)

**Вычисление метрик**

from sklearn.metrics import accuracy\_score, classification\_report, RocCurveDisplay, ConfusionMatrixDisplay, confusion\_matrix

In [25]:

y\_true **=** 20 **\*** ['Education'] **+** 20 **\*** ['Art'] **+** 20 **\*** ['Sport'] **+** 20 **\*** ['Chemical'] **+** 20 **\*** ['Space']

predictions **=** pd**.**concat([preds['Education'], preds['Art'], preds['Sport'], preds['Chemical'], preds['Space']], ignore\_index**=**True)

print(classification\_report(y\_true, predictions))

plt**.**figure(figsize**=**(8, 5))

cm **=** confusion\_matrix(y\_true, predictions, labels**=**['Education', 'Art', 'Sport', 'Chemical', 'Space'])

sns**.**heatmap(cm, annot**=True**, fmt**=**'d', cmap**=**'viridis',

xticklabels**=**preds**.**columns,

yticklabels**=**preds**.**columns

)

plt**.**xlabel('Predicted')

plt**.**ylabel('True')

plt**.**title('Confusion Matrix');

from sklearn.metrics import accuracy\_score

print(f"Accuracy: {accuracy\_score(y\_true, predictions):.2f}")

Accuracy: 0.91

## **Приложение 2**

import os

os**.**environ['HF\_HOME'] **=** 'D:\\HuggingFace'

os**.**environ['TRANSFORMERS\_CACHE'] **=** os**.**environ['HF\_HOME']

os**.**environ['HUGGINGFACE\_HUB\_CACHE'] **=** os**.**environ['HF\_HOME']

from transformers import pipeline

def get\_pred(query):

answer **=** pipe(query, classes, multi\_label**=**False)

return [answer['labels'], answer['scores']]

def chunk\_text(text, window\_size**=**3, overlap**=**1):

sentences **=** sent\_tokenize(text)

chunks **=** []

for i in range(0, len(sentences), window\_size-overlap):

chunks.append(" ".join(sentences[i:i+window\_size]))

return chunks

def aggregate(document):

*# Делим документ*

chunks **=** chunk\_text(document)

*# Предсказываем каждую часть*

preds **=** []

for chun in chunks:

preds.append(get\_pred(chun))

*# Считаем взвешенную метрику*

scores = defaultdict(float)

for pred in preds:

for label, score in zip(pred[0], pred[1]):

scores[label] += score

return max(scores, key=scores.get)

import re

def process\_info(file):

with open(file, "rb") as f:

reader = PyPDF2.PdfReader(f)

text = ''.join([page.extract\_text() for page in reader**.**pages])

*# Служебная информация*

text **=** re**.**sub(r'ISSN\s+\d{4}-\d{3,4}[^\n]\*', '', text)

text **=** re**.**sub(r'\d{4};\d{2}\(\d+\):\d+–\d+', '', text)

*# Авторы*

text **=** re**.**sub(r'[А-ЯЁ][а-яё]+\s+[А-ЯЁ][\.\s]+\s\*[А-ЯЁ][\.\s]\*', '', text)

text **=** re**.**sub(r'[А-ЯЁ][а-яё]+\s+[А-ЯЁ][а-яё]+\s+[А-ЯЁ][\.\s]+\s\*[А-ЯЁ][\.\s]\*', '', text)

text **=** re**.**sub(r'\d+[\s\w\.,–-]+(университет|институт|академия|центр)[^\n]\*', '', text)

*# Сноски в квадратных скобках*

text **=** re**.**sub(r'\[\d+\]', '', text) *# [1], [2]*

text **=** re**.**sub(r'\[\d+[,-]\d+\]', '', text) *# [1-3], [4,5]*

text **=** re**.**sub(r'\[[A-Za-z]+\d\*\]', '', text) *# [A1], [B]*

*# email*

text **=** re**.**sub(r'\S+@\S+', '', text)

*# английские разделы*

text **=** re**.**sub(r'Abstract[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

text **=** re**.**sub(r'Keywords[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

text **=** re**.**sub(r'For citation[^\n]\*[\s\S]\*?(?=\n[А-ЯЁ]|$)', '', text)

*# ссылки*

text **=** re**.**sub(r'http[s]?://(?:[a-zA-Z]|[0-9]|[$-\_@.&+]|[!\*\\(\\),]|(?:%[0-9a-fA-F][0-9a-fA-F]))+', '', text)

text **=** re**.**sub(r'DOI:\s\*\S+', '', text)

*# библиография*

text **=** re**.**sub(r'Список\s+источников[\s\S]\*', '', text)

text **=** re**.**sub(r'References[\s\S]\*', '', text)

*# спец.символы*

text **=** text**.**replace('\xa0', ' ')**.**replace('•', '')

text **=** re**.**sub(r'-\s+', '', text)

text **=** re**.**sub(r'\s+', ' ', text)**.**strip()

text **=** re**.**sub(r'•\s\*\n', '', text)

*# оставшиеся английские фрагменты*

text **=** re**.**sub(r'(?:[A-Za-z-]+\s){3,}[A-Za-z-]\*', '', text)

return text

from transformers import pipeline

import psycopg2

from psycopg2 import sql, errors

from collections import defaultdict

from nltk import sent\_tokenize

import PyPDF2

import pathlib

DB\_CONFIG = {

"dbname": "Electronic\_library",

"user": "postgres",

"password": "Turbo",

"host": "localhost",

"port": "5432"

}

pipe = pipeline(

"zero-shot-classification",

model="MoritzLaurer/mDeBERTa-v3-base-mnli-xnli",

hypothesis\_template="Определи тему научной статьи '{}'",

disable\_hf\_device\_logging=True,

)

classes = ['Education', 'Art', 'Sport', 'Chemical', 'Space']

def create\_connection():

"""Устанавливает соединение с PostgreSQL."""

try:

conn = psycopg2.connect(\*\*DB\_CONFIG)

print("✅ Подключение к PostgreSQL успешно")

return conn

except Exception as e:

print(f"❌ Ошибка подключения: {e}")

return None

def create\_table(conn):

"""Создает таблицу для хранения классифицированных статей."""

create\_table\_query = """

CREATE TABLE IF NOT EXISTS articles (

id SERIAL PRIMARY KEY,

url TEXT NOT NULL UNIQUE,

education BOOLEAN DEFAULT FALSE,

art BOOLEAN DEFAULT FALSE,

sport BOOLEAN DEFAULT FALSE,

chemical BOOLEAN DEFAULT FALSE,

space BOOLEAN DEFAULT FALSE

);

"""

try:

with conn.cursor() as cursor:

cursor.execute(create\_table\_query)

conn.commit()

print("✅ Таблица 'articles' создана или уже существует")

except Exception as e:

print(f"❌ Ошибка создания таблицы: {e}")

conn.rollback()

def predict\_class(text):

"""Определяет класс текста."""

try:

result = aggregate(text)

return result

except Exception as e:

print(f"❌ Ошибка классификации: {e}")

return None

def insert\_article(conn, url, text):

"""Добавляет статью в БД, устанавливая True в нужный столбец, остальные NULL."""

predicted\_class = predict\_class(text)

if not predicted\_class:

return False

try:

with conn.cursor() as cursor:

*# Создаем словарь для обновления (все NULL, кроме predicted\_class)*

data = {class\_name.lower(): (True if class\_name == predicted\_class else None)

for class\_name in classes}

data['url'] = url

*# Формируем SQL-запрос*

columns = sql.SQL(', ').join(map(sql.Identifier, data.keys()))

values = sql.SQL(', ').join([sql.Placeholder()] \* len(data))

query = sql.SQL("""

INSERT INTO articles ({columns})

VALUES ({values})

ON CONFLICT (url) DO UPDATE SET

{update\_clause}

""").format(

columns=columns,

values=values,

update\_clause=sql.SQL(', ').join([

sql.SQL("{} = EXCLUDED.{}").format(

sql.Identifier(k),

sql.Identifier(k)

) for k in data.keys() if k != 'url'

])

)

cursor.execute(query, list(data.values()))

conn.commit()

print(f"✅ Статья '{url}' добавлена в класс '{predicted\_class}'")

return True

except Exception as e:

print(f"❌ Ошибка вставки: {e}")

conn.rollback()

return False

if \_\_name\_\_ == "\_\_main\_\_":

conn = create\_connection()

if conn:

create\_table(conn)

insert\_article(

conn,

url="https://elibrary.ru/download/elibrary\_26738028\_39746529.pdf",

text=process\_info("art.pdf")

)

conn.close()