# Проект Система классификации и рекомендации научных статей

# Выполняли: Кондаков Андрей и Яценко Артём

**Постановка задачи**

1. Обработка pdf документов, содержащих научные статьи. Выделение заголовка, описания (abstract) и параграфов статьи
2. Поиск ключевых слов в статье. Поиск статей, содержащих определённые ключевые слова
3. Рекомендация наиболее похожих статей по теме
4. Если статья на медицинскую тему, то обучить NER модель для детекции названий болезней в статье.

**Работа с API сайта arxiv.org, создание базы статей**

*Над задачей работал Яценко Артём*

С помощью API arxiv.org удалость обработать ~40000 статей. Для каждой статьи я скачивал её название, описание, ссылку на .pdf файл, ссылку на страницу статьи на arxiv.

Для хранения статей была развёрнута облачная база mongoDB на бесплатном сервисе mongoDB Atlas. Описания статей хранятся в базе без препроцессинга (вместе с LaTeX-формулами, для возможности отправки полноценного текста пользователю).

Перед использованием в моделях текст проходит следующий препроцессинг: очистка от формул и небуквенных символов, фильтрация стоп-слов, токенизация, лемматизация посредством WordNetLemmatizer (nltk) с использованием POS-тэгов.

**Реализация и анализ алгоритмов для поиска ключевых слов в научных статьях**

*Над задачей работали Кондаков Андрей, Яценко Артём*

Для поиска ключевых слов в текстах нами рассматривались 3 различных алгоритма:

1. Выделение как ключевые слова с наибольшим Tf-idf
2. RAKE (Rapid Automatic Keywords extraction)
3. TextRank

Сравнение эффективности каждого из алгоритмов тестировалось на корпусе новостных статей CNN (<https://www.kaggle.com/harishcscode/all-news-articles-from-home-page-media-house>). Для оценки использовались точность (precision), полнота (recall) и f-мера для каждого метода. После анализа приведённой ниже таблицы и сравнения тех ключевых слов, которые генерируют методы на выбранных научных статьях с сайта arxiv.org, было принято решение использовать первый метод.

*Артём:*

Для фильтрации ключевых слов, полученных после TF-IDF я также использовал POS-тэги. То есть ключевые слова (с учётом того, что мы используем N-граммы до 2-х слов) должны следовать следующему паттерну: последнее слово N-граммы — существительное, предыдущие слова — прилагательные/герундий/существительные и т.п. Это помогло получить намного более релевантные ключевые слова.

**Поиск статей в базе по ключевым словам**

*Над задачей работал Яценко Артём*

В описанной ранее базе mongoDB я создал отдельную коллекцию с ключевыми словами для статей и настроил соответствующий поисковый индекс. Запрос к базе считает для каждой статьи количество совпадений с заданными ключевыми словами, сортирует результаты на основе этой информации.

**Парсинг статей из формата pdf, извлечение текста Abstract из статьи**

*Над задачей работал Кондаков Андрей*

Для извлечения текста из PDF статей была использована библиотека PDFminer которая извлекает тест пдф статьи в специальный временный текстовый файл. Позже из этого файла извлекается текст Abstract а статьи, для извлечения я решил использовать не сложное регулярное выражение, его использование обоснованно тем что большинство статей на сайте arxiv.org имеют схожую структуру где блок Abstract находится между словами “ Abstract” и “Contents” хотя и есть небольшое количество статей с уникальной структурой, но данный метод работает быстро и вполне стабильно

**Поиск похожих статей по базе**

*Над задачей работали Кондаков Андрей, Яценко Артём*

*Андрей:*

Для поиска статей со схожим описанием я решил считать косинусное расстояние между векторизованными представлениями текстов Abstract статей. Для векторизации я использовал модель Text2Vec из библиотеки genism. Данную модель я предварительно обучил на статьях из базы скаченной с arxiv.org. С помощью данной модели была создана таблица embedding-ов статей (размерность векторов embedding была выбрана равной 100).

*Артём*:

Так как задача поиска наиболее похожей статьи по эмбеддингу — задача поиска ближайшего соседа, то для увеличения быстродействия и хранения эмбеддингов, я решил использовать векторную базу данных milvus.  
Нахождение похожих статей проходит по индексу IVF\_FLAT (разбиение исходного множества точек на кластеры, поиск расстояния до центров кластеров).

**Поиск названий болезней в медицинском тексте**

*Над задачей работал Кондаков Андрей*

Дляобучения модели по предсказанию является ли слово названием болезни был использован корпус размеченных медицинских статей (<https://huggingface.co/datasets/ncbi_disease>)

Метод который я использовал для предсказания “Тагов” слов схож с решением задачи POS-Tagging. Перед обучением все тексты были предварительно тонизированы. По тонизированным текстам я составил словарь символов, сопоставив каждому символу соответствующий номер.

Позже по был составлена таблица где каждый текст датасета был представлен таблицей (Размер самого длинного текста, Размер самого длинного токена). Первый слой моей модели представлял собой Char Embedding который с помощью одномерного сверточного слоя обучал эмбадинги для каждого символа слова. Сначала я остановился на подобной модели просто считая эмбадинг каждого слова с помощью MaxPooling а потом пропуская полученную таблицу (Размер самого длинного текста, Размер эмбадинга) через линейный слой. Но подобная модель не учитывала соседние слова при классификации. Поэтому я добавил ещё один свёрточный слой который работал уже с таблицей эмбэдингов слов.

Описание Модели:

SentenceDiseaseRecognizer(

(char\_embeddings): Embedding(63, 32, padding\_idx=0)

(single\_token\_backbone): StackedConv1d(

(layers): ModuleList(

(0): Sequential(

(0): Conv1d(32, 32, kernel\_size=(3,), stride=(1,), padding=(1,))

(1): Dropout(p=0.3, inplace=False)

(2): LeakyReLU(negative\_slope=0.01)

)

)

)

(context\_backbone): StackedConv1d(

(layers): ModuleList(

(0): Sequential(

(0): Conv1d(32, 32, kernel\_size=(3,), stride=(1,), padding=(1,))

(1): Dropout(p=0.3, inplace=False)

(2): LeakyReLU(negative\_slope=0.01)

)

)

)

(global\_pooling): AdaptiveMaxPool1d(output\_size=1)

(out): Conv1d(32, 3, kernel\_size=(1,), stride=(1,))

)

Тагже я попытался визуализировать модель с использованием библиотеки

(<https://github.com/waleedka/hiddenlayer>)

Полученное изображение модели прикрепил в гитхаб репозитории проекта.

После обучения я проверил точность модели с помощью функции classification\_report из библиотеки sklearn сверяя точность модели в взвешенному среднему f меры по всем классам (0.72) (модель для каждого слова предсказывает один из 3 классов: обычное слово, название болезни состоящее из одного слова, часть многосложного названия)

Пример работы модели:

Germline BRCA1 alterations in a population-based series of ovarian cancer cases. -> Germline-<NOTAG> BRCA1-<NOTAG> alterations-<NOTAG> in-<NOTAG> a-<NOTAG> population-<NOTAG> based-<NOTAG> series-<NOTAG> of-<NOTAG> ovarian-<PNE> cancer-<PNE> cases-<NOTAG>

**Создание серверной части**

*Над задачей работал Кондаков Андрей*

Для создания серверной части было написано простое веб приложение с помощью библиотеки Flask для каждой из вышеописанных задач был описан соответствующий запрос

**Развёртывание сервера**

*Над задачей работал Яценко Артём*

В качестве платформы для сервера я выбрал Okteto. Важна была возможность развёртывать приложение с помощью Kubernetes, потому что это облегчает установку milvus (есть готовый helm-чарт).

Для деплоя на Okteto я сделал docker-образ нашего проекта.

**Ссылки**

Ссылка на Github: <https://github.com/AndreyKondakovGW/NLP-Case/tree/main>

Ссылка на Googlecolab notebook в котором Андрей обучал NER модель:

<https://colab.research.google.com/drive/10w89avTuWGWw9ZlMszPEGBL3tlzHNv2K>