МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ
Кафедра дискретной математики и алгоритмики

ЖАРИН Глеб Вадимович

ПРЕДСКАЗАНИЕ КОДА УДК ПО ТЕКСТУ ДОКУМЕНТА

Магистерская диссертация

специальность 1-31 80 09 "Прикладная математика и информатика"

Научный руководитель
Гецевич Юрий Станиславович
кандидат технических наук

Допущена к защите

«__» _____ 2021 г.

Зав. Кафедрой дискретной математики и алгоритмики
_____ В.М. Котов
доктор физико-математических наук, профессор

Минск, 2021

ОГЛАВЛЕНИЕ

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Магистерская диссертация, 60 с., 22 рис., 7 таблиц, 31 источник, 4 приложения.

Ключевые слова: АВТОМАТИЧЕСКАЯ КЛАССИФИКАЦИЯ, УНИВЕРСАЛЬНАЯ ДЕСЯТИЧНАЯ КЛАССИФИКАЦИЯ, ИНТЕЛЛЕКТУАЛЬНЫЙ АНАЛИЗ ТЕКСТОВ, ИНТЕЛЛЕКТУАЛЬНАЯ ОБРАБОТКА ТЕКСТОВ.

Объект исследования – коды Универсальной десятичной классификации (УДК) для документов на белорусском, русском и английском языках.

Цель работы – разработать и реализовать алгоритм автоматической классификации документов по системе УДК.

Методы проведения работы – эксперименты в области машинного обучения и обработки текстовых данных.

Результаты – разработанный и реализованный алгоритм автоматической классификации документов по системе УДК в виде общедоступного сервиса в сети Интернет.

Область применения – библиотечное дело, интеллектуальный анализ текстов.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Магістарская дысертацыя, 60 с., 22 мал., 7 табліц, 31 крыніца, 4 дадатка.

Ключавыя словы: АЎТАМАТЫЧНАЯ КЛАСІФІКАЦЫЯ, УНІВЕРСАЛЬНАЯ ДЗЕСЯТКОВАЯ КЛАСІФІКАЦЫЯ, ІНТЭЛЕКТУАЛЬНЫ АНАЛІЗ ТЭКСТАЎ, ІНТЭЛЕКТУАЛЬНАЯ АПРАЦОЎКА ТЭКСТАЎ.

Аб’ект даследавання – коды Універсальнай дзесятковай класіфікацыі (УДК) для дакументаў на беларускай, рускай і англійскай мовах.

Мэта работы – распрацаваць і рэалізаваць алгарытм аўтаматычнай класіфікацыі дакументаў па сістэме УДК.

Метады правядзення працы – эксперыменты ў вобласці машыннага навучання і апрацоўкі тэкставых дадзеных.

Вынікі – распрацаваны і рэалізаваны алгарытм аўтаматычнай класіфікацыі дакументаў па сістэме УДК у выглядзе агульнадаступнага сэрвісу ў сетцы Інтэрнэт.

Вобласць прымянення – бібліятэчная справа, інтэлэктуальны аналіз тэкстаў.

ABSTRACT

Master’s thesis, 60 p., 22 fig., 7 tables, 31 sources, 4 appendices.

Keywords: AUTOMATIC CLASSIFICATION, UNIVERSAL DECIMAL CLASSIFICATION, TEXT MINING, TEXT PROCESSING.

Object of research – Universal decimal classification (UDC) codes for documents in Belarusian, Russian and English.

Objective – develop and implement an automatic document classification algorithm according to the UDC system.

Methods – experiments in machine learning and text processing.

Results – an algorithm for automatic document classification according to the UDC system was developed and implemented in the form of a publicly available service on the Internet.

Application area – library management, text mining.

ВВЕДЕНИЕ

Магистерская работа посвящена автоматизации процесса классификации документов по Универсальной десятичной классификации (далее, УДК).

Под документом в системе УДК может пониматься любой источник информации, в том числе книги, журналы, статьи, музыкальные произведения, фильмы и т.п.. В настоящей работе для автоматической классификации используется текстовая информация о документе, так что описанным способом при наличии соответствующего набора данных можно классифицировать любой тип документов. Однако основное внимание уделяется собранному в ходе работы набору данных информации о книгах на белорусском, русском и английском языках.

УДК – одна из самых популярных в мире многоязычных систем классификации для всех областей знаний и сложный инструмент индексации и поиска. Особенно популярна эта система в библиотечном деле стран СНГ, в том числе библиотеках Республики Беларусь.

В соответствии с системой УДК каждый классифицируемый документ получает специальный код, который в дальнейшем можно использовать для его поиска.

Благодаря своей логической иерархической структуре и аналитико-синтетической природе, УДК подходит для физической организации коллекций, а также для просмотра и поиска документов. УДК структурирована таким образом, чтобы можно было легко включать новые разработки и новые области знаний. Сам код УДК не зависит от какого-либо конкретного языка или алфавита (состоит из арабских цифр и общих знаков препинания), поэтому сопутствующие описания классов появились для многих языков мира.

Однако из-за своей своей универсальности проставление кода УДК документу (например, новой книге) является нетривиальной задачей и требует времени специально обученных людей (библиотекарей). Этим обусловлена актуальность темы – автоматически получать код УДК для нового документа.

Объектом исследования работы являются коды УДК для документов на белорусском, русском и английском языках, а целью – разработка и реализация алгоритма автоматической классификации документов по системе УДК.

Для достижения цели работы были произведены теоретические исследования системы УДК (глава 1), выполнен автоматизированный поиск, сбор, анализ и обработка набора данных кодов УДК (глава 2). На основе полученного набора данных были обучены модели автоматического предсказания кодов УДК документов. Описание обучения моделей, их оценка, также их применение при разработке общедоступного веб-сервиса на платформе «Corpus.by» [1] описаны в главе 3.

В приложениях А, Б, В и Г приведены соответственно исходные коды программ автоматизированного сбора, анализа, извлечения и подготовки набора данных, чтобы любой желающий мог воспроизвести результаты работы.

ГЛАВА 1
ОБЩИЕ СВЕДЕНИЯ О СИСТЕМЕ УДК

1.1 Основные положения УДК

Универсальная десятичная классификация (УДК) – это язык индексации документов в виде схемы классификации, охватывающей всю совокупность знаний. УДК предназначен для тематического описания и индексации содержания информационных ресурсов независимо от носителя, формы, формата или языка.

Центральной частью УДК являются таблицы, охватывающие всю совокупность знаний и построенные по иерархическому принципу деления от общего к частному с использованием цифровых десятичных кодов [2].

Каждая последующая цифра десятичного кода уточняет класс документа, деля его на не более, чем десять подклассов. Такой «десятичный принцип» и дал название классификации – «Универсальная десятичная классификация».

Система УДК состоит из следующих частей: основная таблица, вспомогательные таблицы и алфавитно-предметные указатели [3].

Основная таблица описывает конкретные области знаний и представляет собой соотношения «код УДК – описание класса документа».

Первая цифра кода определяет наиболее общие области знаний:

0 Наука в целом
1 Философия. Психология
2 Религия. Теология
3 Социальные науки
4 Зарезервированный на будущее класс
5 Математика. Естественные науки
6 Прикладные науки. Медицина. Техника
7 Искусство. Декоративно-прикладное искусство. Фотография. Музыка. Игры. Спорт
8 Языкознание. Филология. Художественная литература. Литературоведение
9 География. Биология. История

Последующие цифры уточняют класс документа. Например:

5 Математика. Естественные науки
51 Математика
510 Фундаментальные и общие проблемы математики

Пример кода УДК для документа: *299.8*. В соответствии с таблицами можно расшифровать этот код как следующий класс: *2* – Религия. Теология; *29* – Нехристианские религии; *299* – Прочие религии; *299.8* – Религии Южной

Америки. Религии доколумбовских цивилизаций (караибских народов, инков и др.).

Вспомогательные таблицы нужны для определения правил создания комбинированных классов и описания специальных знаков, которые могут использоваться в кодах УДК помимо цифр. Например, в них описывается ряд соединительных символов, позволяющих связывать и расширять номера УДК (основные перечислены в таблице 1).

Символ	Значение	Пример
+	Согласование, присоединение	*59+636* – Зоология и Животноводство
/	Последовательное продление	*592/599* – Систематическая зоология (т.е. все классы от *592* до *599*)
:	Отношение	*17:7* – Взаимоотношение Этики и Искусства
[]	Группировка	*311:[622+669]* – Статистика Горного дела и Металлургии (здесь знаки относятся к группе в [] как к целому)
*	Обозначение не из УДК	*523.4*433* – Планетология, малая планета Эрос (после * номер планеты)
A/Z	Алфавитное указание	*929NAP1* – Биография Наполеона I Бонапарта

Таблица 1 – Основные соединительные символы

Помимо соединительных символов используются так называемые «определители»:

=... – определитель языка;
(0...) – определитель формы: учебник, реферат, статья и т.п.;
(1/9) – определитель места;
(=...) – определитель рас, народов, национальностей;
“...” – определитель времени;
-... – другие общие определители: свойств, материалов и лиц.

Алфавитно-предметный указатель – семантический справочник по тексту таблиц УДК, содержащий перечень основных понятий, отсортированных по алфавиту, с указанием соответствующего кода класса.

В настоящее время система УДК содержит десятки тысяч классов и переведена на почти шестьдесят языков, что делает её лидером по числу охватываемых областей знаний по сравнению с другими существующими классификационными системами.

Более подробное описание системы классификации можно найти на официальном сайте консорциума УДК – некоммерческой ассоциации издателей, созданной для поддержки, развития и распространения УДК в интересах пользователей [4].

1.2 Выводы

В этой главе были рассмотрены основные теоретические принципы системы УДК.

Можно заключить, что коды УДК имеют нетривиальную структуру, состоящую из множества символов, связанных между собой различными отношениями. Это порождает десятки тысяч уникальных классов для классификаторов, не считая возможных алфавитных указаний, обозначений не относящихся к системе УДК, определителей времени, места и т.п..

Отсюда можно сделать вывод, что вручную извлечь всевозможные отношения и взаимосвязи в коде УДК для более-менее точной классификации является сложной задачей. В связи с этим был выбран подход на основе машинного обучения с учителем, который заключается в автоматическом нахождении необходимых связей с помощью большого количества примеров.

В следующей 2 главе рассмотрим получение, анализ и обработку набора данных таких примеров.

В 3 главе рассмотрим сам алгоритм машинного обучения, оценим получившиеся модели и применим обученные классификаторы для создания программного комплекса для автоматической классификации по системе УДК.

ГЛАВА 2
СБОР, АНАЛИЗ И ОБРАБОТКА НАБОРА ДАННЫХ

2.1 Сбор набора данных кодов УДК

При рассмотрении немногочисленных существующих исследований по теме (например, [5]), автору не удалось найти в них готовых наборов данных, удовлетворяющих всем следующим требованиям:

- в качестве данных выступают тексты книг (открытая лицензия на использование авторской литературы);
- данные представлены белорусским, русским и английским языками (мультиязычность);
- качественная библиотечная разметка кодами УДК (коды УДК проставлены профессиональными библиотекарями).

Поэтому было решено самостоятельно собрать набор данных, несколько смягчив первое требование – в качестве данных выступает общедоступная метаинформация о книге (заголовок, язык написания и т.п.).

Поставленным требованиям удовлетворяет электронный каталог Национальной библиотеки Республики Беларусь [6]: в открытом доступе находится множество записей о книгах на белорусском, русском и английском языках, большинству из которых проставлены коды УДК профессиональными библиотекарями.

Для поиска документа в каталоге доступны следующие поисковые поля:

- тип;
- автор;
- темы;
- жанры;
- серии;
- издательства;
- языки;
- дата публикации.

Для составления трёх наборов данных (по одному для каждого языка) был написан Python-скрипт для автоматического скачивания результатов поиска в электронном каталоге с использованием библиотек Requests и BeautifulSoup [7, 8]. Исходный код скрипта приведён в Приложении А.

Скрипт позволяет задать множество параметров поиска, сохранить результаты и продолжить закачку с произвольного места, восстановить работоспособность при потере интернет-соединения, делать компрессию и декомпрессию данных.

Для настройки параметров скрипта используется интерфейс командной строки, помощь к которому представлена на рисунках 2.1, 2.2 и 2.3.

```
usage: nlb_crawler.py [-h] data_file log_file [start_page] [end_page] [--option]...

The National Library of Belarus E-Catalogue crawler.

Files:
  Arguments related to files.

  data_file             the file path for the collected data (type: FileType('wb'), the file is opened only after the --prev_data_file reading and closing to support read
                        and write to the same file)
  log_file              the file path for logging (type: FileType('a'))
  --prev_data_file FILE
                        the file path for the previously collected data to concatenate with the current data (type: FileType('rb'), the file must be in pickle format,
                        optionally compressed by gzip, bz2, lzma, zip or lz4 compression standard (the compression standard is inferred from the path's extension))
  --compression {gzip,bz2,lzma,zip,lz4,none}, -c {gzip,bz2,lzma,zip,lz4,none}
                        the compression standard for the collected data (type: str, default: lz4)

Pages:
  Arguments related to the results pages.

  start_page            the first page to process (type: positive int, default: 1)
  end_page              the last page to process (type: positive int, default: all available pages)
  --limit {20,50,100}, -l {20,50,100}
                        the number of documents per page (type: int, default: 100)
```

Рисунок 2.1 – Параметры скрипта для скачивания набора данных (часть 1)

```
Query:
  Arguments related to the query string.

  --lookfor QUERY, -q QUERY
                        the query string to search for in the document field (type: str, default: any query)
  --type {AllFields,Title,Author,Subject,ISN}, -t {AllFields,Title,Author,Subject,ISN}
                        the document field in which to search for the query string (type: str, default: AllFields)
  --sort {relevance,year,year+asc,author,title}, -s {relevance,year,year+asc,author,title}
                        the documents sort order by the query string (type: str, default: relevance)

Filters:
  Arguments related to the search filters.

  --national_doc        search only for The Belarusian National Document(type: bool, default: False)
  --format {Book,AuthorDissertation,Article,MusicalDocumentation,Dissertation,Image,Serial_Continue,Serial,Rare,AudioVideo,Cartographic,InternationalOrg,Manuscripts,Electron
ic,Standardization,Reviews,Scientifically}
                        search only for this document format/type (type: str, default: Book)
  --author_sort AUTHOR, --author AUTHOR
                        search only for this main author documents (type: str, default: any author)
  --topic_facet TOPIC [TOPIC ...], --topic TOPIC [TOPIC ...]
                        search only for documents with these topics (type: str, default: any topic)
  --genre_facet GENRE [GENRE ...], --genre GENRE [GENRE ...]
                        search only for documents with these genres (type: str, default: any genre)
```

Рисунок 2.2 – Параметры скрипта для скачивания набора данных (часть 2)

```
  --series_facet SERIES [SERIES ...], --series SERIES [SERIES ...]
                        search only for documents that are parts of these series (type: str, default: any series)
  --publishPlace PLACE [PLACE ...], --place PLACE [PLACE ...]
                        search only for documents published in these places (type: str, default: any place)
  --language LANG [LANG ...], --lang LANG [LANG ...]
                        search only for documents containing all of these languages (type: str, default: any language)
  --publishDatefrom YEAR, --from YEAR
                        search only for documents published from this year (type: int in [1400; 2022], default: 1400)
  --publishDateto YEAR, --to YEAR
                        search only for documents published by this year (type: int in [1400; 2022], default: 2022)

Other:
  Other arguments.

  --help, -h            show this help message and exit
  --timeout NUM         the number of seconds to wait for the server to send data before giving up (type: non-negative float, default: 30.0)
  --retries NUM, -r NUM
                        the number of retries after a failed request (type: non-negative int, default: 5)
```

Рисунок 2.3 – Параметры скрипта для скачивания набора данных (часть 3)

Помимо самого кода УДК скрипт сохраняет и всю метаинформацию о документе, представленную в формате BibTeX [9] в электронном каталоге.

BibTeX-файл состоит из записей следующего вида:

```
@ARTICLE{any_tag,
  author = {Список авторов},
  title = {Название статьи},
  year = {Год},
  journal = {Название журнала},
  udk = {Код УДК документа, если присутствует}
}
```

Здесь *ARTICLE* — тип записи («статья»), *any_tag* – метка-идентификатор записи (которая позволяет ссылаться на неё в тексте с помощью *\cite{any_tag}*), дальше – список полей со значениями.

Также скрипт ведёт лог, показывающий текущий прогресс закачки и возникающие ошибки. Согласно ему, при закачке белорусского набора данных ошибок при парсинге BibTeX-записей не возникало, в то время как для английского языка возникло 4 ошибки, а для русского – 10. При использованных настройках закачки каждая ошибка означает потерю 100 документов для набора данных.

Для скачивания белорусского набора данных понадобилось около 2 часов, английского – около 3,5 часов, а русского – целая неделя.

2.2 Анализ собранных данных

Чтобы понять, какие алгоритмы классификации можно использовать для собранного набора данных, необходимо проанализировать полученные записи. Также этап анализа поможет понять, как правильно очистить данные (например, убрать ошибки разметки) и выбрать подмножество данных для обучения (например, использовать для ускорения обучения только наиболее богатые классы УДК).

Для анализа собранных в прошлом пункте данных был написан ещё один Python-скрипт с использованием библиотеки Matplotlib [10]. Его исходный код приведён в Приложении Б.

Скрипт позволяет вывести множество статистических характеристик собранных данных и построить гистограммы распределения данных, получая описания классов УДК из кодов с помощью веб-сервиса «UDC Decoder» [11].

Для настройки параметров этого скрипта, как и прошлого, используется интерфейс командной строки (см. рисунок 2.4).

```
usage: nlb_analyzer.py [-h] data_file log_file bar_plots_dir [--option]...

The National Library of Belarus E-Catalogue data analyzer.

Files:
  Arguments related to files.

  data_file           the file path for the collected data (type: FileType('rb'))
  log_file            the file path for logging (type: FileType('a'))
  bar_plots_dir       the dir path for bar plots (type: Path)

Other:
  Other arguments.

  --help, -h          show this help message and exit
  --timeout NUM       the number of seconds to wait for the server to send data before giving up (type: non-negative float, default: 10.0)
  --retries NUM, -r NUM
                      the number of retries after a failed request (type: non-negative int, default: 1)
```

Рисунок 2.4 – Параметры скрипта для анализа набора данных

Рассмотрим подробнее полученные данные.

В таблице 2 представлены основные количественные характеристики белорусского, русского и английского собранных наборов данных: общее количество скачанных данных, количество записей с указанным кодом УДК, количество записей с уникальными кодами УДК.

	Белорусский	Русский	Английский
Всего записей	68’796	2’589’069	123’811
Записей с известным кодом УДК	65’090	1’869’802	102’987
Уникальных кодов УДК	28’079	904’872	56’371

Таблица 2 – Количественные характеристики скачанных наборов данных

Из неё можно сделать вывод, что белорусский набор данных самый маленький, однако имеет более высокий процент размеченных данных. Английский набор данных примерно в 2 раза больше, а русский – примерно в 35, что объясняет разницу во времени закачки.

Также мы сразу же можем сделать вывод, что применить наивный алгоритм мультиклассовой классификации [12] вида «бинарный классификатор каждый-с-каждым» не получится из-за большого количества уникальных классов. Поэтому для уменьшения количества классов была применена предобработка, описанная в пункте 2.3, идеи для которой были получены в результате следующего анализа.

Во-первых, интересен вопрос, действительно ли все документы соответствуют своему языку (русский заголовок документа для записи из русского набора данных и т.д.).

При поиске в электронном каталоге указание языка означает лишь наличие этого языка в списке языков документа, но не гарантирует, что заголовок или другая метаинформация будет на этом языке. Поэтому для определения языка заголовка была использована предобученная на статьях с Википедии, а также ещё на двух корпусах [13, 14], модель для автоматического определения языка текста [15].

На рисунках 2.5 и 2.6 представлены, соответственно, гистограмма распределения языков в белорусском наборе данных с точки зрения указания в записях электронного каталога и с точки зрения определения заголовков документов с помощью вышеуказанной модели.

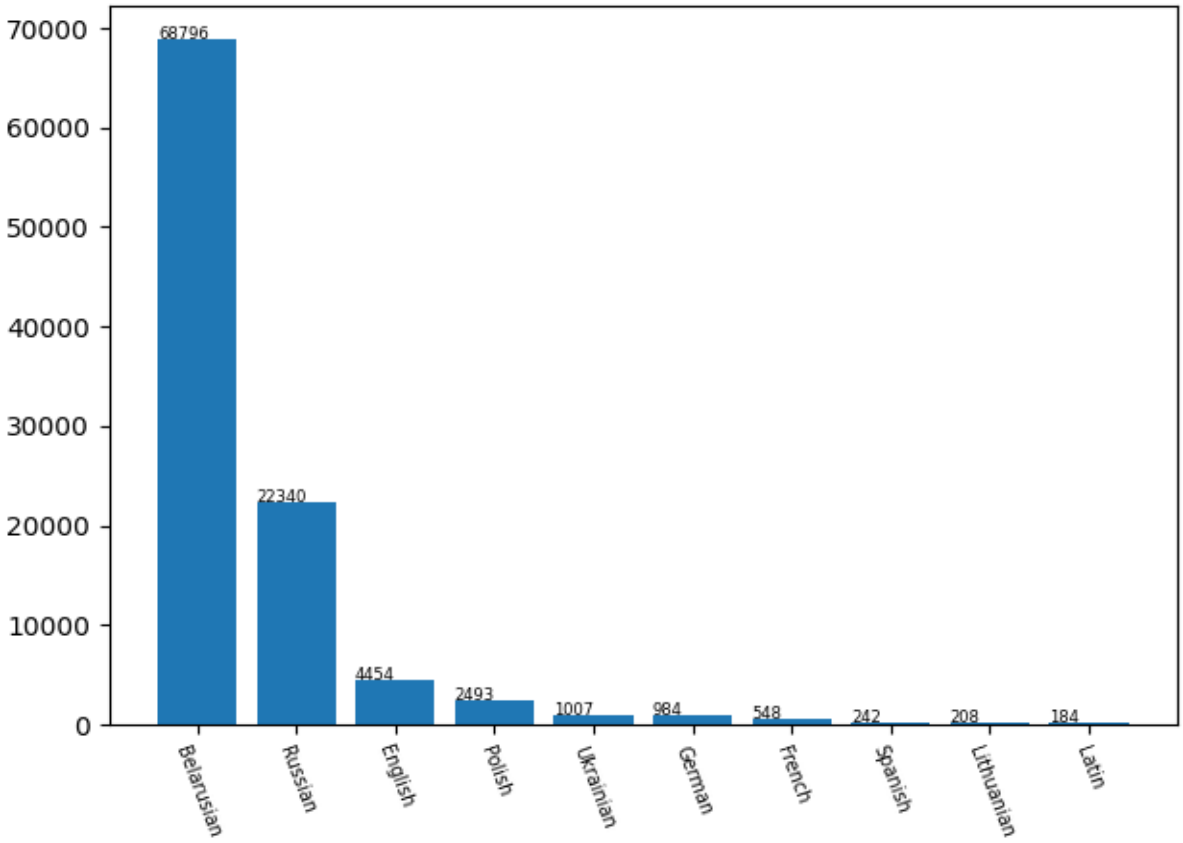

Рисунок 2.5 – Распределение языков в записях белорусского набора данных

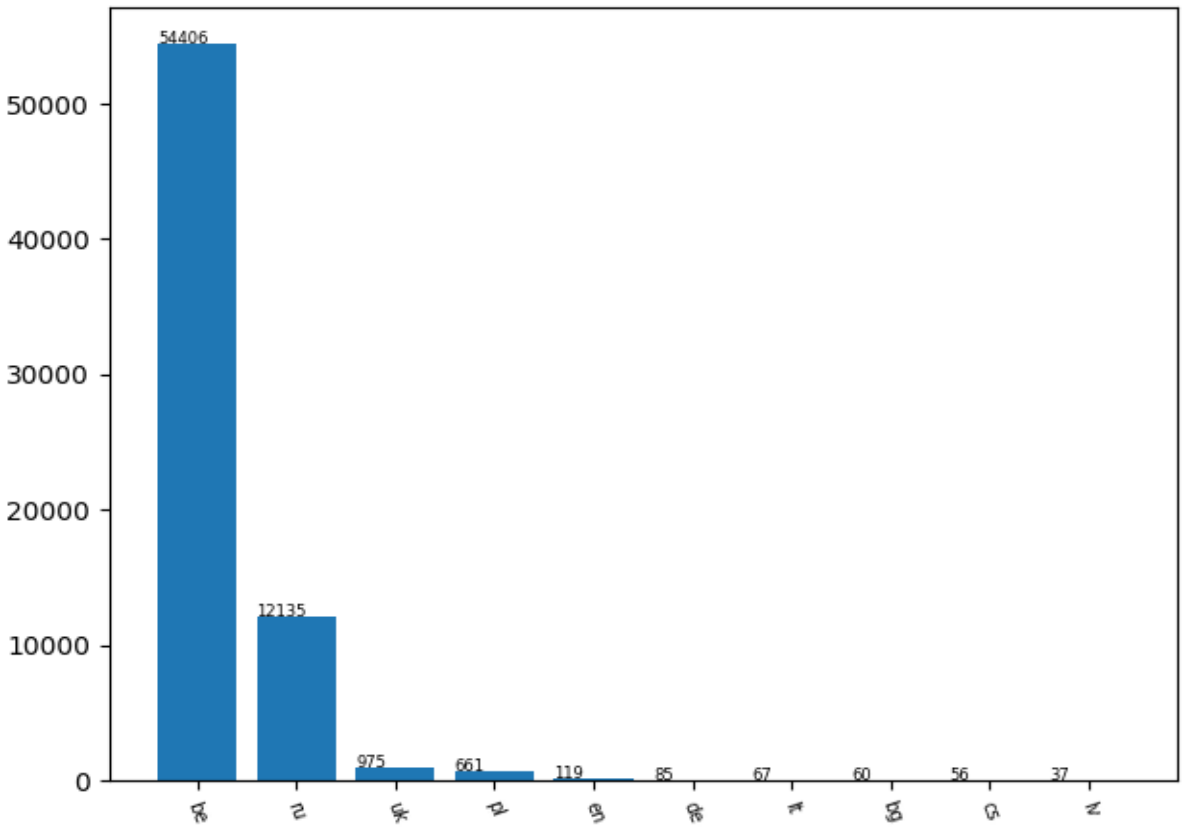

Рисунок 2.6 – Распределение языков в заголовках белорусского набора данных

Видно, что не все документы с указанным в записи наличием белорусского языка имеют заголовки на белорусском языке. Аналогичная картина и для русского с английским наборов данных (рисунки не приведены для краткости).

Во-вторых, интересен вопрос распределения данных по классам УДК или насколько сбалансированы собранные данные.

На рисунках 2.7, 2.8, 2.9 представлены гистограммы распределений кодов УДК белорусского, русского и английского наборов данных по основным классам УДК (первой цифре кода).

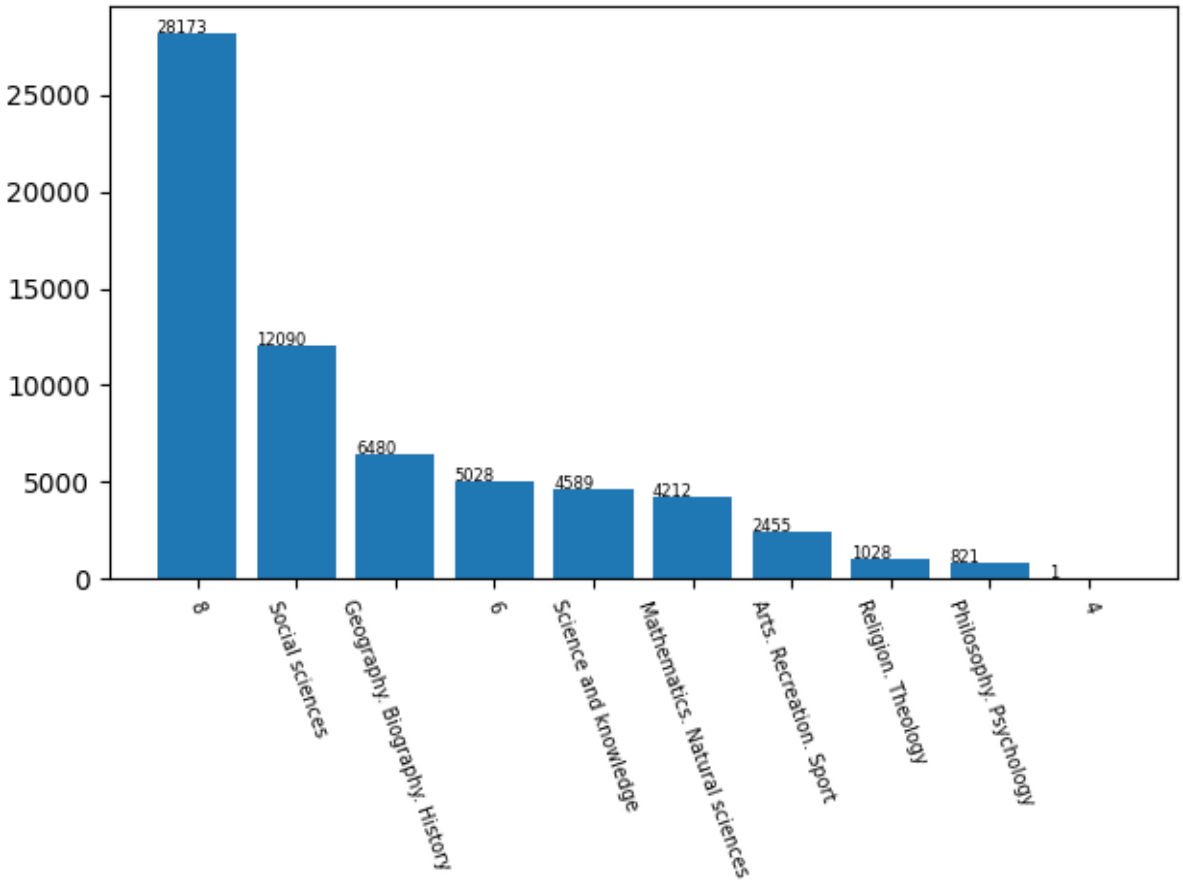

Рисунок 2.7 – Распределение данных по основным классам УДК для белорусского набора данных

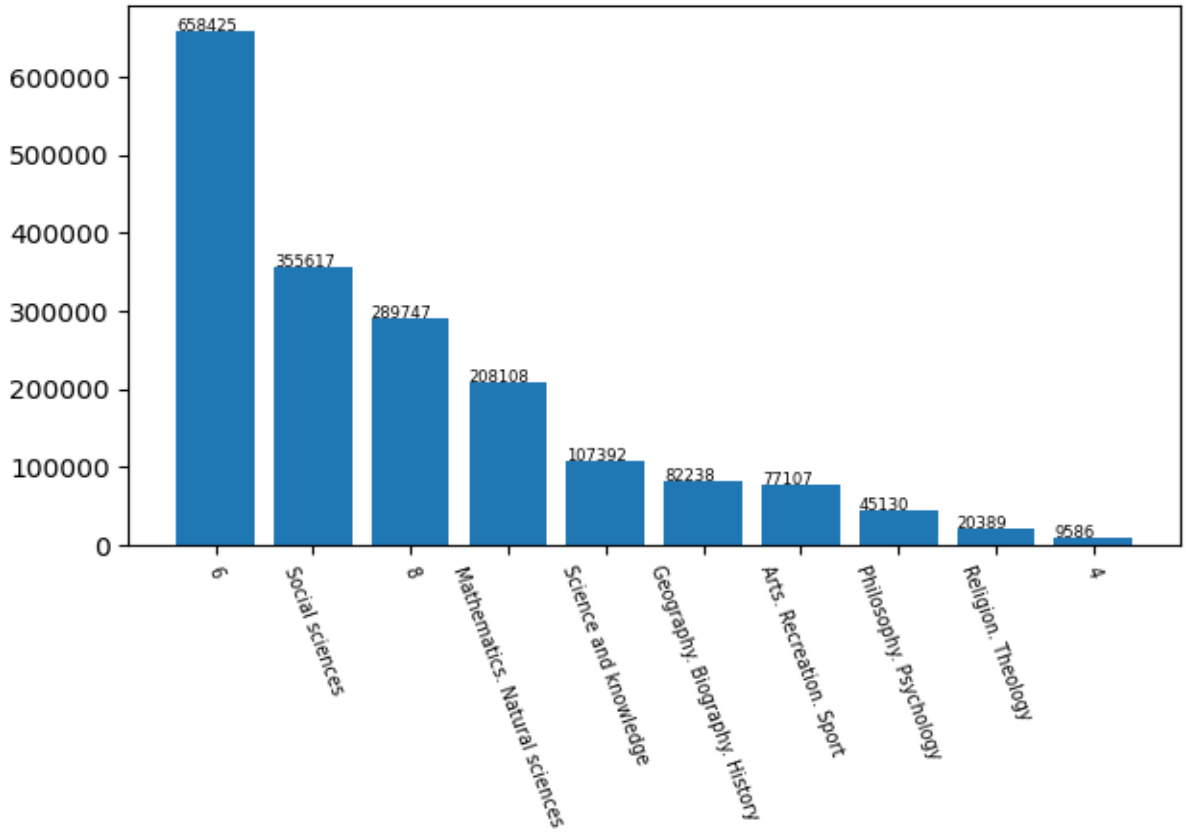

Рисунок 2.8 – Распределение данных по основным классам УДК для русского набора данных

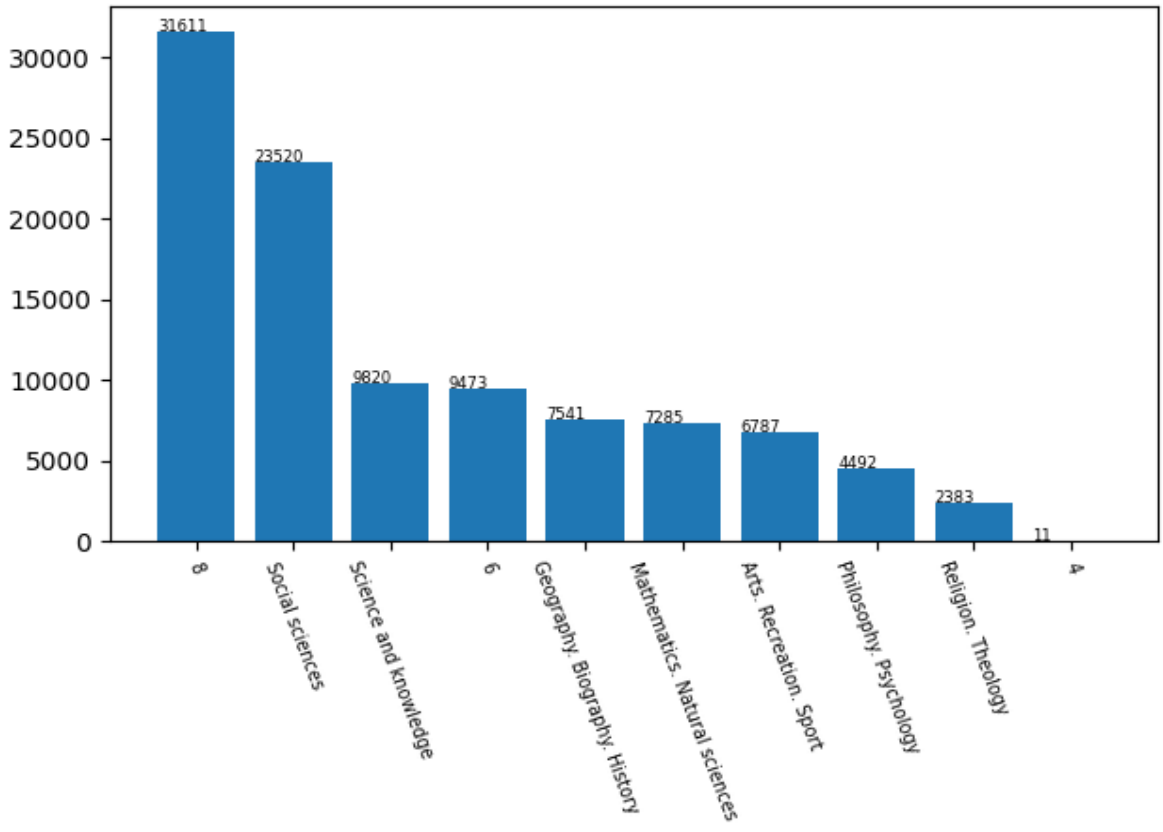

Рисунок 2.9 – Распределение данных по основным классам УДК для английского набора данных

По графикам можно сделать вывод, что белорусский и английский наборы данных содержат в основном художественную литературу, а русский – литературу по прикладным наукам.

Также замечаем, что в данных присутствуют коды УДК, начинающиеся на *4*, а, как мы знаем из главы 1, такие коды являются зарезервированными, то есть имеет место ошибка разметки либо намеренное проставление незнакомым классам этого кода.

Также отмечаем сильную несбалансированность классов во всех трёх наборах данных, что также накладывает сложности на алгоритм классификации.

В-третьих, рассмотрим какие вообще метаданные представлены в записях электронного каталога. Рассмотрим их распределение на примере русского набора данных (см. рис. 2.10).

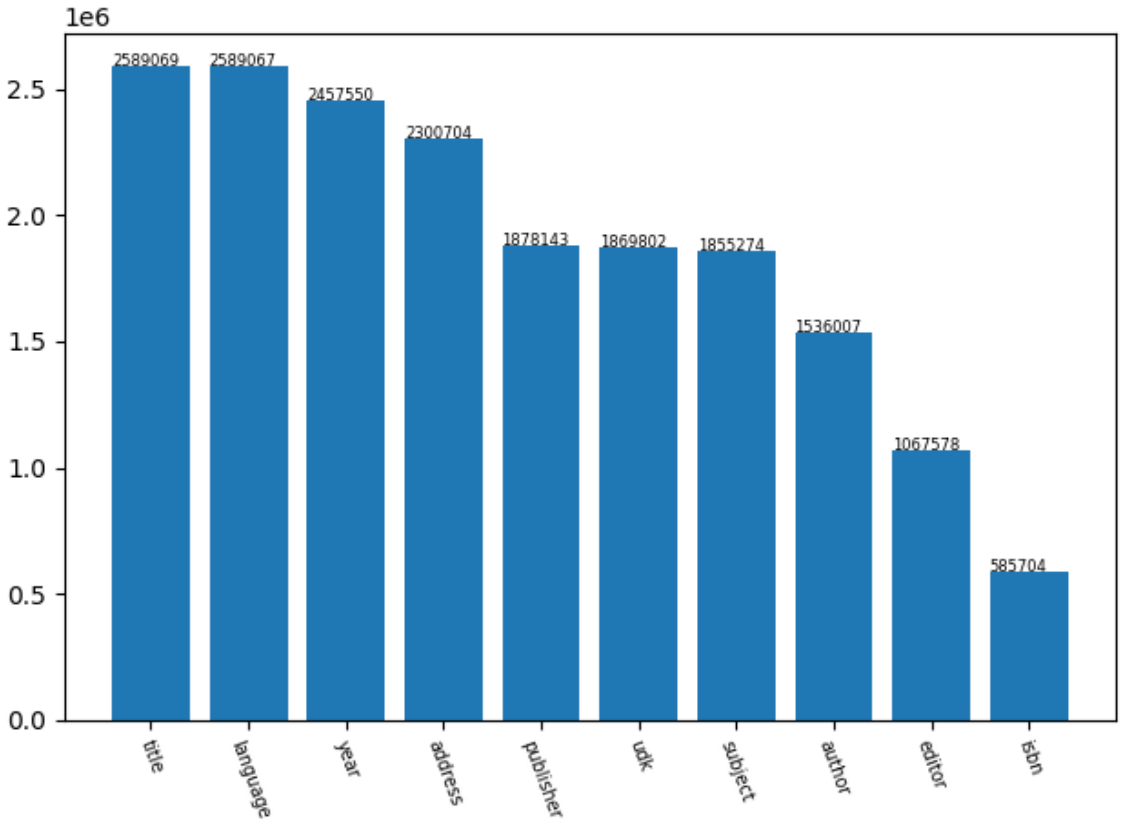

Рисунок 2.10 – Распределение метаданных в записях русского набора данных

Видим, что наиболее частыми помимо заголовка и языка являются данные о годе и адресе документа. Эти данные могут быть использованы для тренировки классификаторов определителей времени и места (см. главу 1).

Интуитивно полезные данные об авторе документа содержатся не во всех записях с указанным УДК, но в большинстве из них.

Крайне полезные данные о теме документа содержатся практически в стольких же документах, в скольких указан код УДК (однако, это не значит, что пересечение велико).

Наконец, рассмотрим, насколько можно упростить наши наборы данных, если просто оставить коды УДК, состоящие из цифр и точек (без специальных символов и определителей, описанных в 1 главе). Эту информацию иллюстрируют рисунки 2.11, 2.12, 2.13.

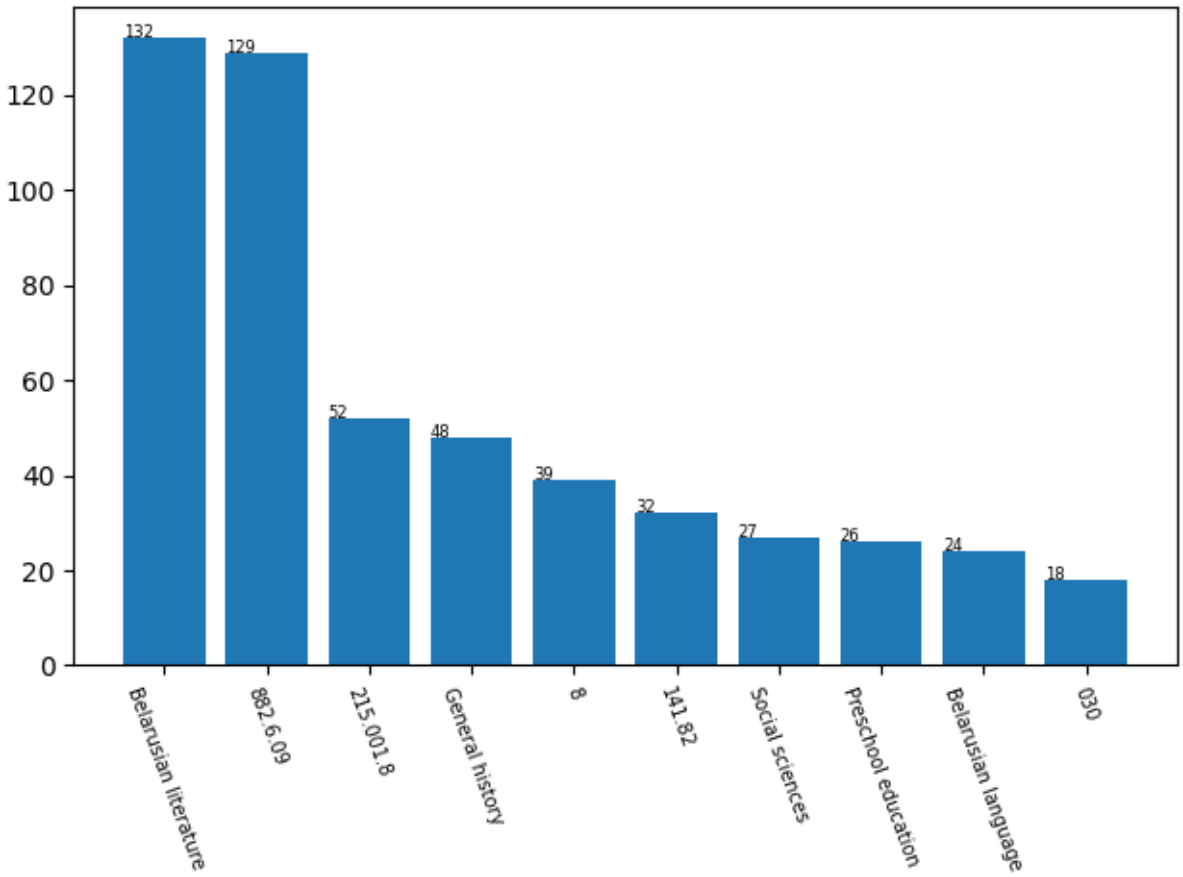

Рисунок 2.11 – Самые популярные классы, состоящие только из цифр и точек, в белорусском наборе данных

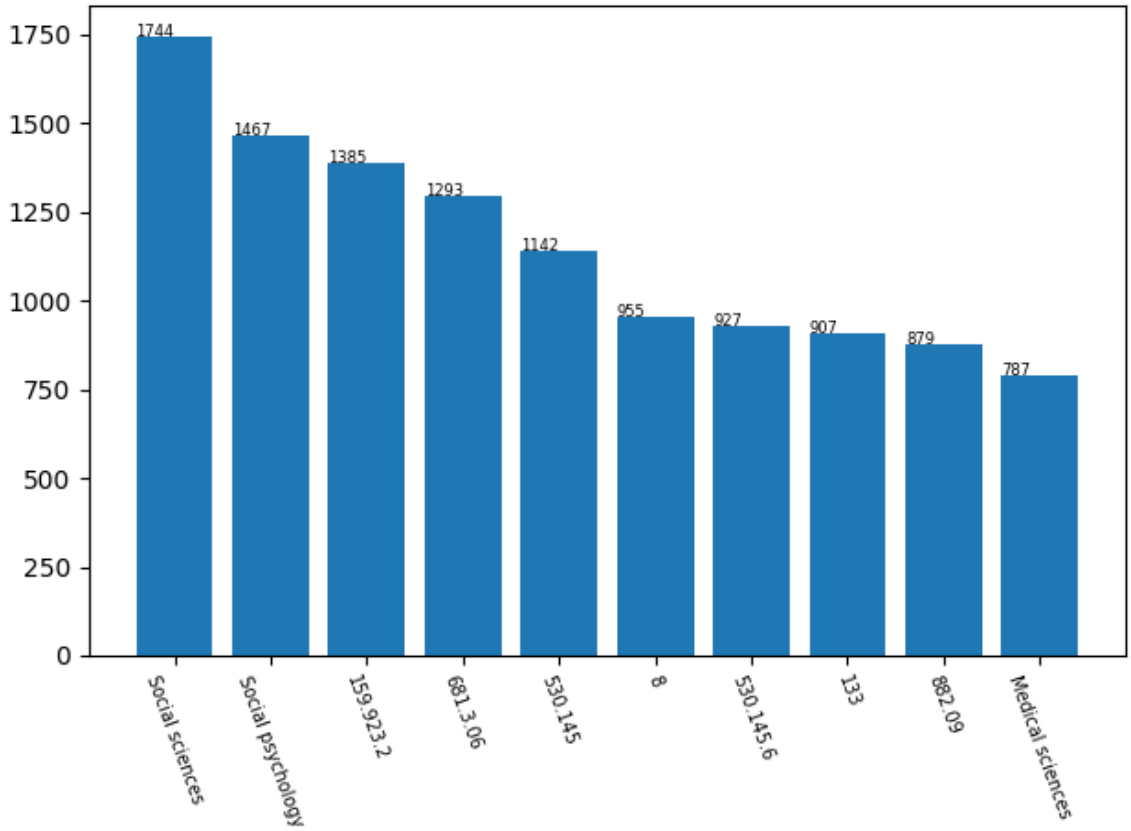

Рисунок 2.12 – Самые популярные классы, состоящие только из цифр и точек, в русском наборе данных

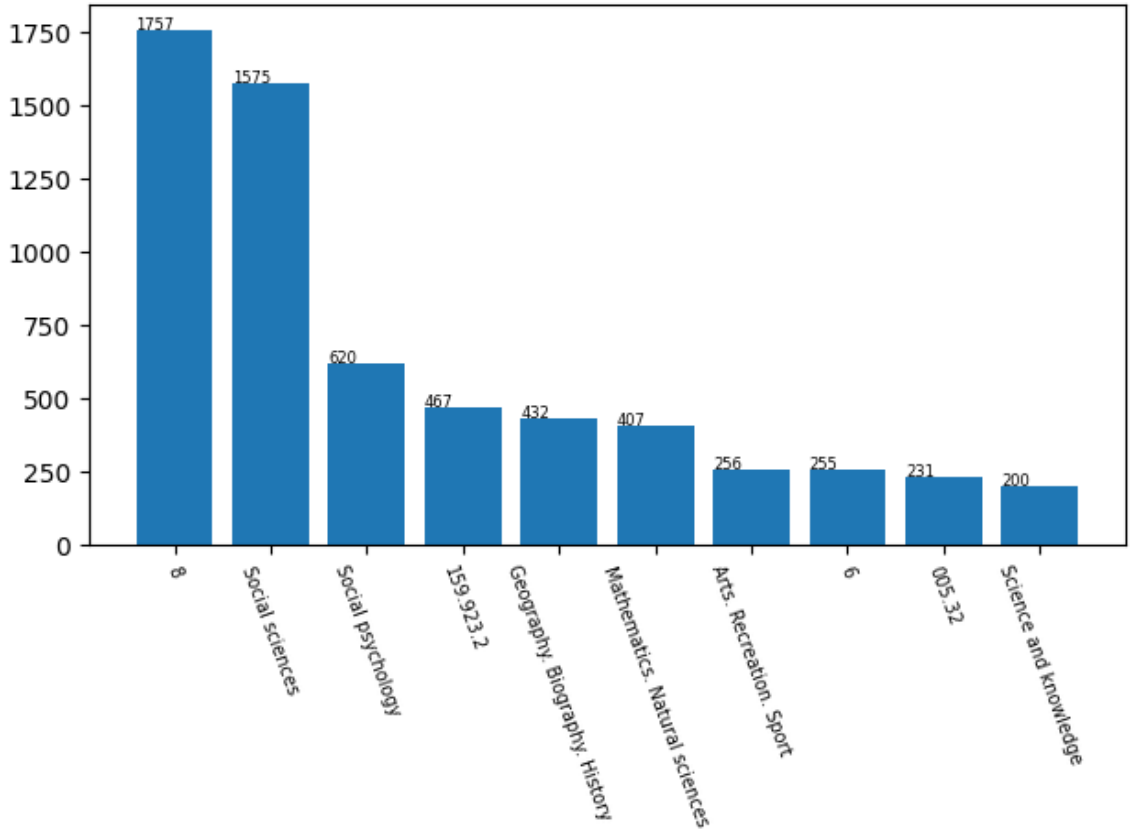

Рисунок 2.13 – Самые популярные классы, состоящие только из цифр и точек, в английском наборе данных

Видно, что идея обучения на таких упрощённых данных не самая лучшая. В белорусском наборе данных их просто мало для обучения, а в русском и английском самые популярные классы связаны друг с другом, то есть классификаторы, скорее всего, всегда будут предсказывать один или два популярных класса в наборе данных.

Помимо распределений данных были собраны различные характеристики как кодов УДК, так и заголовков книг, которые в пункте 2.3 будут использоваться для их очистки. Например, для кодов УДК интересны частота использования различных символов и наличие/отсутствие определителей времени и формы. Для заголовков интересны специальные символы, разделяющие непосредственно заголовок документа от метаинформации документа (обычно это знак «слеш»).

Пример собранных характеристик для белорусского набора данных приведен на рисунках 2.14 и 2.15.

```
lang in title = Counter({'be': 54406, 'ru': 12135, 'uk': 975, 'pl': 661, 'en': 119, 'de': 85, 'lt': 67, 'bg': 60, 'cs': 56, 'lv': 37, 'et': 28, 'sr': 26, 'es': 24, 'sk': 24,
 'fr': 19, 'hu': 14, 'ky': 8, 'ro': 7, 'sl': 7, 'sv': 5, 'it': 3, 'ce': 3, 'eo': 3, 'tt': 3, 'nl': 3, 'kk': 2, 'mn': 2, 'cv': 2, 'hr': 2, 'tg': 2, 'fi': 2, 'sh': 1, 'gl': 1,
 'ar': 1, 'vi': 1, 'tr': 1, 'az': 1})
contain "/" in title = 63756
contain " / " in title = 63695
contain "-" in title = 62756
contain " - " in title = 60714
contain "–" in title = 10225
contain " – " in title = 8969
doesn't contain "/", "-", "–" in title = 4
```

Рисунок 2.14 – Различные характеристики кодов УДК и заголовков из белорусского набора данных (часть 1)

```
UDC code specified = 65090
unique UDC code = 28079
code symbols = {'ё', 'ж', 'Z', 'H', 'e', ',', ' ', 'l', 'D', 'Ё', 'в', ':', 'i', 'Т', '1', 'б', 'č', 'В', 'ł', 'Д', 'О', 'Ч', '6', 'Р', 'э', 'm', 'щ', 'М', 'W', 'р', 'м', 't
', 'ы', 'i', 'ч', 's', 'о', 'а', '(', 'о', 'ю', 'ц', 'т', 'У', 'п', 'Ш', '2', 'w', '0', '7', 'ш', 'у', 'š', 'Е', 'я', 'К', 'Н', 'Б', 'Г', 'В', 'Е', '_', 'j', 'e', 'I', 'k',
'д', '/', 'н', 'a', 'G', 'З', 'р', 'М', '8', 'А', 'v', '9', 'А', 'у', 'Э', 'И', 'Ł', 'ў', 'г', 'b', 'Р', 'с', 'х', '=', 'Х', 'С', '{', 'd', 'П', 'ф', 'и', 'ū', '+', '[', 'I'
, 'h', '.', 'з', 'Т', 'Я', 'Ś', 'Ю', ']', 'к', 'И', 'Ф', '4', 'й', 'g', 'ъ', "'", 'Ж', 'S', 'r', 'Ц', 'N', 'ь', 'u', 'V', 'J', 'c', '}', '<', 'n', '3', 'F', '"', 'z', 'Л', '
’', '-', 'л', 'C', '5', 'K', 'O', '>', 'R', '\xa0', ')', 'ń', 'á', 'L'}
non-alphanumeric symbols = {'=', '{', ',', ' ', '}', ':', '<', '[', '+', '"', '.', "'", '_', '-', '/', ']', '>', '\xa0', ')', "'", '('}
plain digits and dots = 1933
unique plain digits and dots = 865
contain "." = 58534
contain "+" = 3584
contain "and" = 18166
contain "and" surrounded by whitespace = 18166
contain "/" = 5796
contain ":" = 7661
contain "::" = 0
contain "*" = 0
contain lang det = 2121
contain form det = 33610
contain place det = 25646
contain time det = 3480
```

Рисунок 2.15 – Различные характеристики кодов УДК и заголовков из белорусского набора данных (часть 2)

2.3 Предобработка данных

Исходя из анализа данных (см. пункт 2.2), было принято решение в качестве данных использовать очищенные от метаинформации заголовки книг соответствующего языка, а в качестве меток – очищенные от определителей, неизвестных символов и ошибок коды УДК.

Для извлечения нужных метаданных из записей (заголовок и код УДК), а также объединения записей с заголовками одного языка из разных наборов данных, был написан новый Python-скрипт. Его исходный код приведён в Приложении В.

Скрипт позволяет задать различные ключи, которые надо извлекать из записей, язык, которому должен соответствовать заголовок (определяется так же, как в пункте 2.2), а также способ сжатия данных для хранения на диске.

Интерфейс командной строки для настройки скрипта значительно отличается от интерфейсов предыдущих скриптов (см. рисунок 2.16).

```
usage: nlb_data_extractor.py [-h] data_file... [--option]...

The National Library of Belarus E-Catalogue data extractor.

Files:
  Arguments related to files.

  data_files           file paths for the collected data (type: FileType('rb'))
  --extracted_data_file DIR, -o DIR
                       the file path for the extracted data (type: Path, default: data.pickle.lz4)
  --compression {gzip,bz2,lzma,zip,lz4,none}, -c {gzip,bz2,lzma,zip,lz4,none}
                       the compression standard for the extracted data (type: str, default: lz4)

Data:
  Arguments related to the data extraction.

  --keys KEY [KEY ...], -k KEY [KEY ...]
                       extract data with these keys, "title" key is required if --language is not 'any' (type: str, default: ['title', 'udk'])
  --language LANG, -l LANG
                       extract data with this language in title (type: str, default: any)

Other:
  Other arguments.

  --help, -h           show this help message and exit
```

Рисунок 2.16 – Параметры скрипта для извлечения нужных данных из записей

После извлечения необходимых ключей из записей набора данных заголовок и код УДК необходимо очистить для уменьшения размера пространства данных и меток. Далее уже можно составлять готовый для обучения набор данных с разделением на тренировочное и тестовое подмножества.

Для этих целей предобработки был написан Python-скрипт с использованием библиотек spaCy, datrie и regex [16, 17, 18]. Его исходный код приведён в Приложении Г.

Скрипт позволяет выбирать ключи, которые и будут после очистки представлены классификаторам. Также есть возможность выбрать не все данные, а только данные из N самых популярных классов, а также указать пропорцию деления данных на тренировочную и тестовую выборки.

Интерфейс командной строки для настройки скрипта приведен на рисунке 2.17.

```
usage: nlb_extracted_data_preprocessor.py [-h] data_file... [--option]...

The National Library of Belarus E-Catalogue data preprocessor.

Files:
  Arguments related to files.

  extracted_data_file   the file path for the extracted data (type: FileType('rb'))
  preprocessed_data_dir
                        the dir path for the preprocessed data (type: Path)

Data:
  Arguments related to the data preprocessing.

  --data_key KEY, -X KEY
                        the extracted data key for training (data/X) (type: str, default: title)
  --label_key KEY, -Y KEY
                        the extracted data key for training (label/Y) (type: str, default: udk)
  --top NUM, -t NUM     extract only the top NUM labels (type: int in [0; +inf], default: 100)
  --drop FLOAT, -d FLOAT
                        drop this percent of the top data (type: float in [0; 1], default: 0.0)
  --train_valid_test_split_ratio FLOAT, -r FLOAT
                        split the extracted data into train, validation and test sets with this ratio (the number means train percentage, validation and test are equally
                        splitted from the rest) (type: float in [0; 1], default: 0.7)
  --use_spacy_tok_and_lemma {en,ru,none}, -spacy {en,ru,none}
                        use spaCy tokenizer and lemmatizer for this language model (type: str, default: none)

Other:
  Other arguments.

  --help, -h            show this help message and exit
```

Рисунок 2.17 – Параметры скрипта для предобработки данных

После множества экспериментов алгоритмы очистки данных приняли следующий вид.

Алгоритм очистки заголовка:

1. с помощью регулярных выражений [19] отделяем от заголовка части, стоящие после знаков «/», «-», «—», если эти знаки не находятся в круглых или квадратных скобках; важно каждый знак удалять по одному последовательно, а не все вместе;
2. удаляем некоторые левостоящие знаки препинания (которые не могут идти в начале заголовка) и все пробельные символы;
3. удаляем все правостоящие пробельные символы;
4. если последний символ заголовка не из множества символов «.» «?» «!», то добавляем в конец заголовка точку.

Алгоритм очистки кода УДК:

1. с помощью регулярных выражений в цикле удаляем все определители и прочие подстроки в скобках (список подстрок получен в результате анализа из пункта 2.2) от самых вложенных до скобок верхнего уровня;

2. разделяем код на несколько частей по знаку «+» или слову «and» (его часто используют вместо «+», судя по анализу из пункта 2.2);

3. каждую из частей в свою очередь разделяем по специальным знакам «:» и «::» на более мелкие части;

4. для всех самых маленьких частей выполняем следующую процедуру:

4.1 удаляем все суффиксы после символов «*» «-» «=», включая сами эти символы;

4.2 удаляем все открывающиеся скобки, для которых нет соответствующих закрывающихся, оставшиеся после первого шага либо ошибочно выставленные в коде УДК;

4.3 удаляем все закрывающиеся скобки, для которых нет соответствующих открывающихся, оставшиеся после предыдущих шагов либо ошибочно выставленные в коде УДК;

4.4 удаляем все суффиксы после любых символов кроме цифр, «.», «+», «/», «:», включая сами символы;

4.5 удаляем все левостоящие и правостоящие символы «.», «+», «/», «:»;

5. соединяем разделенные по символам «:» и «::» непустые части с помощью символа «:»;

6. соединяем разделенные по символам «+» и слову «and» непустые части с помощью символа «+»;

Исходя из иерархической структуры кодов УДК, было принято решение проводить классификацию по множеству меток [20]. Если заголовок имеет код УДК, например, *82,* то добавим ему ещё и метку *8*. Для расчёта популярности классов после таких преобразований, а также для описания таксономии меток [21] была использована структура данных префиксное дерево [22], позволяющая быстро получить префиксы для строк, которые в случае кодов УДК будут более общими родительскими классами.

Для извлечения токенов-слов из заголовков в русском и английском наборах данных были использованы предобученные наборы данных библиотеки spaCy [23, 24].

Извлекались только начальные формы слов, которые не присутствуют в списках наиболее употребительных слов (т.н. «стоп-слова»), игнорируя знаки препинания. Все слова приводились в нижний регистр.

Для белорусского языка извлечение слов из заголовков (т.н. «токенизация») проводилась вручную с помощью регулярных выражений из-за отсутствия предобученных моделей. Слова не приводились к начальным формам.

Таким образом, подготовленные данные имеют вид «значимые слова или начальные формы значимых слов в нижнем регистре очищенного заголовка – список очищенных кодов УДК».

2.4 Выводы

В этой главе было рассмотрено создание программного комплекса для автоматического сбора, анализа и предобработки наборов данных кодов УДК для документов на белорусском, русском и английском языках.

В результате выполнения описанных скриптов получены наборы данных вида «значимые слова или начальные формы значимых слов в нижнем регистре очищенного заголовка – список очищенных кодов УДК».

Эти наборы данных будут использованы в следующей 3 главе для машинного обучения автоматических классификаторов заголовков книг по системе УДК.

ГЛАВА 3
ОБУЧЕНИЕ И ПРИМЕНЕНИЕ КЛАССИФИКАТОРОВ

3.1 Описание процесса обучения

Для обучения классификаторов на множестве меток использовался фреймворк NeuralNLP-NeuralClassifier [25]. Общая архитектура нейронной сети представлена на рисунке 3.1.

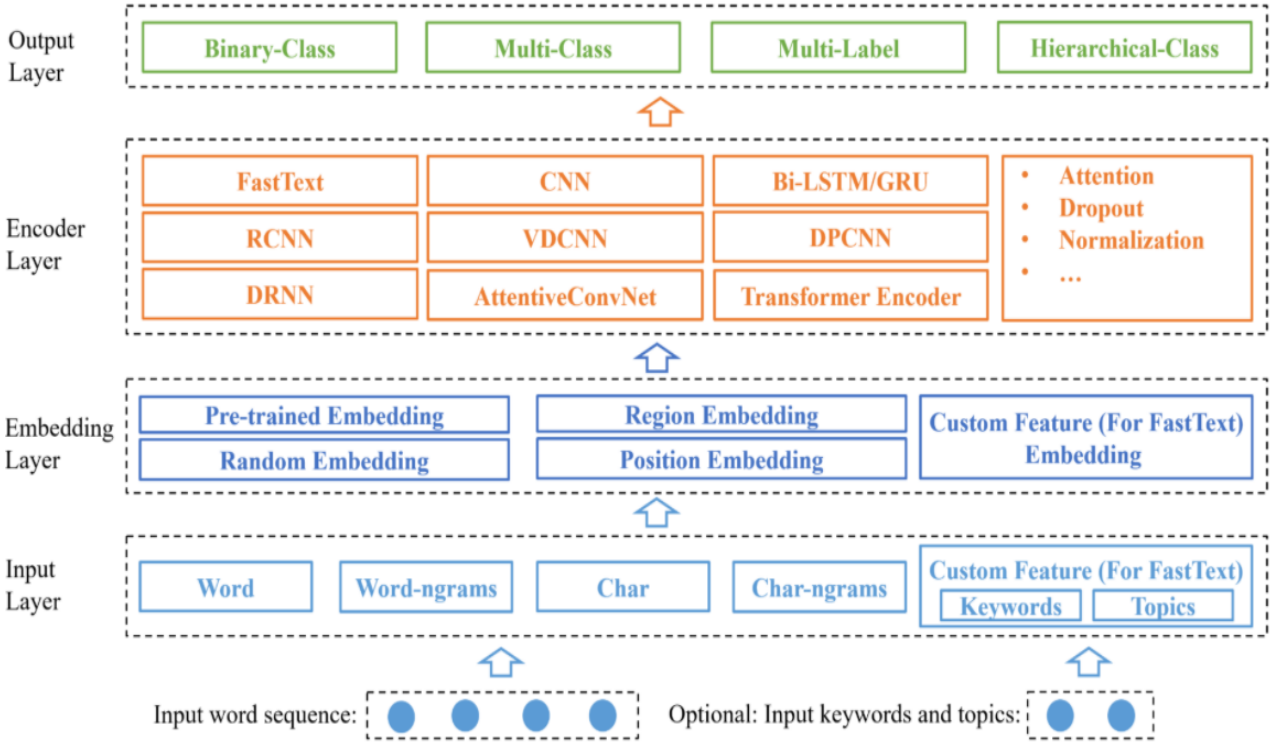

Рис 3.1 – Архитектура нейронной сети фреймворка NeuralClassifier

После различных экспериментов были выбраны параметры для фреймворка, представленные в таблице 3.1.

	Белорусский	Русский	Английский
Input слой	биграммы	униграммы	биграммы
Embedding слой	случайный	случайный	случайный
Output слой	Multi-label	Multi-label	Multi-label
Эпохи	9	7	12
Batch размер	64	64	64

Таблица 3.1 – Параметры фреймворка NeuralClassifier для входных наборов данных

В качестве Encoder слоя или основной обучающей нейронной сети лучше всех себя показала рекуррентная нейронная сеть (RNN) [26], которая и используется в дальнейшем для всех наборов данных.

Использованные параметры для RNN представлены в таблице 3.2. Эти параметры не изменялись, использовались предоставленные по умолчанию фреймворком.

Параметр	**Значение**
Размер скрытого слоя	64
Тип рекуррентной сети	Bidirectional GRU
Количество рекуррентных слоёв	1
Тип Embedding	Attention
Размер Attention слоя	16

Таблица 3.2 – Параметры для основной рекуррентной сети

Везде в качестве функции потерь выступал BCEWithLogitsLoss [27], а в качестве оптимизатора – Adam [28].

Перед началом обучения наборы данных разбивались на тренировочную, валидационную и тестовую выборки в соотношении 0.7, 0.15, 0.15, соответственно.

Характеристики подаваемых на вход фреймворку наборов данных представлены в таблице 3.3.

	Белорусский	Русский	Английский
Уникальные классы	100	100	100
Уникальные токены	45983	132194	24595
Уникальные n-граммы	167543	132194	119927

Таблица 3.3 – Количественные характеристики входных наборов данных

Можно видеть, что русский набор данных был уменьшен во много раз (путём отбрасывания 95% всех данных). Также для уменьшения наборов данных и, соответственно, ускорения обучения, предсказывались только 100 самых популярных меток, а не все возможные метки.

3.2 Оценка результатов обучения

В результате обучения были получены модели с оценками на тренировочной и тестовой выборках, представленными в таблицах 3.4 и 3.5.

	Белорусский	Русский	Английский
Precision	0.963	0.834	0.955
Recall	0.951	0.615	0.934
F-score	0.957	0.708	0.945
Macro F-score	0.930	0.477	0.923
Loss	0.018	0.056	0.021

Таблица 3.4 – Оценка моделей на тренировочных выборках

	Белорусский	Русский	Английский
Precision	0.757	0.755	0.652
Recall	0.617	0.444	0.493
F-score	0.680	0.559	0.562
Macro F-score	0.541	0.213	0.462
Loss	0.079	0.086	0.091

Таблица 3.5 – Оценка моделей на тестовых выборках

Время обучения для белорусского, русского и английского наборов данных составило, соответственно, 15, 61 и 13 минут.

Графики изменения f-score для тестовых выборок белорусского, русского и английского языков представлены, соответственно, на рисунках 3.2, 3.3 и 3.4.

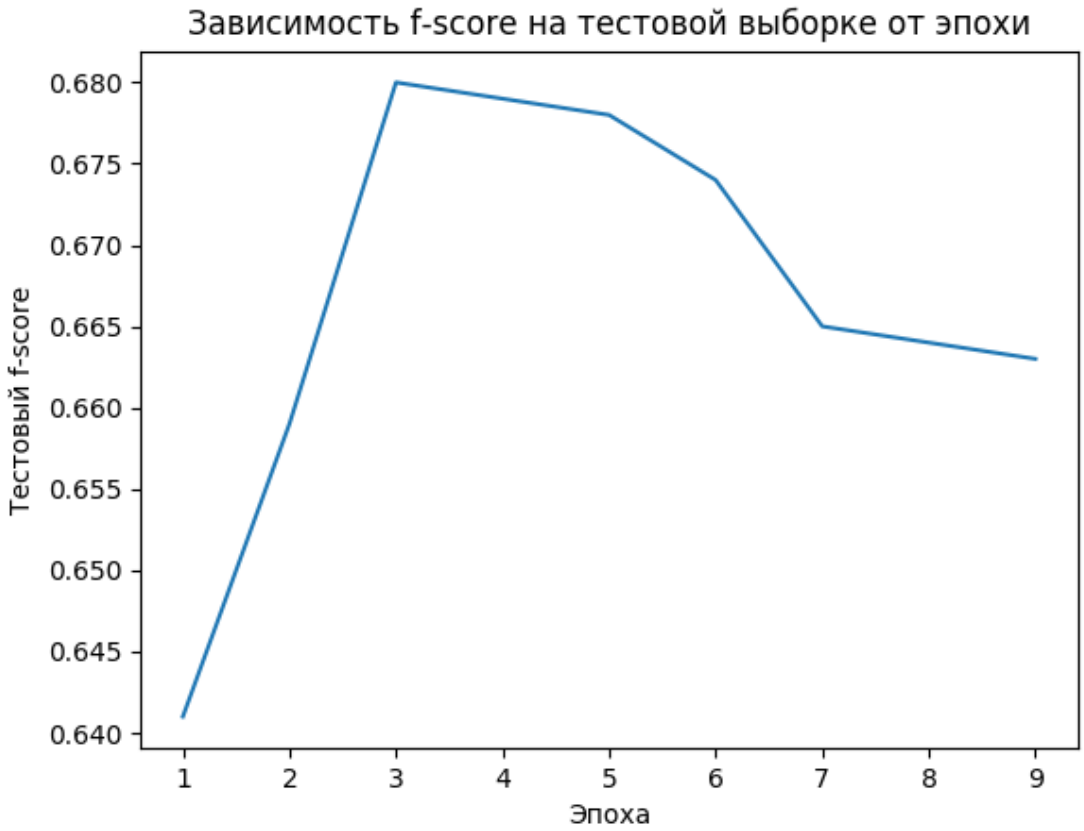

Рисунок 3.2 – Изменение тестового f-score для белорусского набора данных

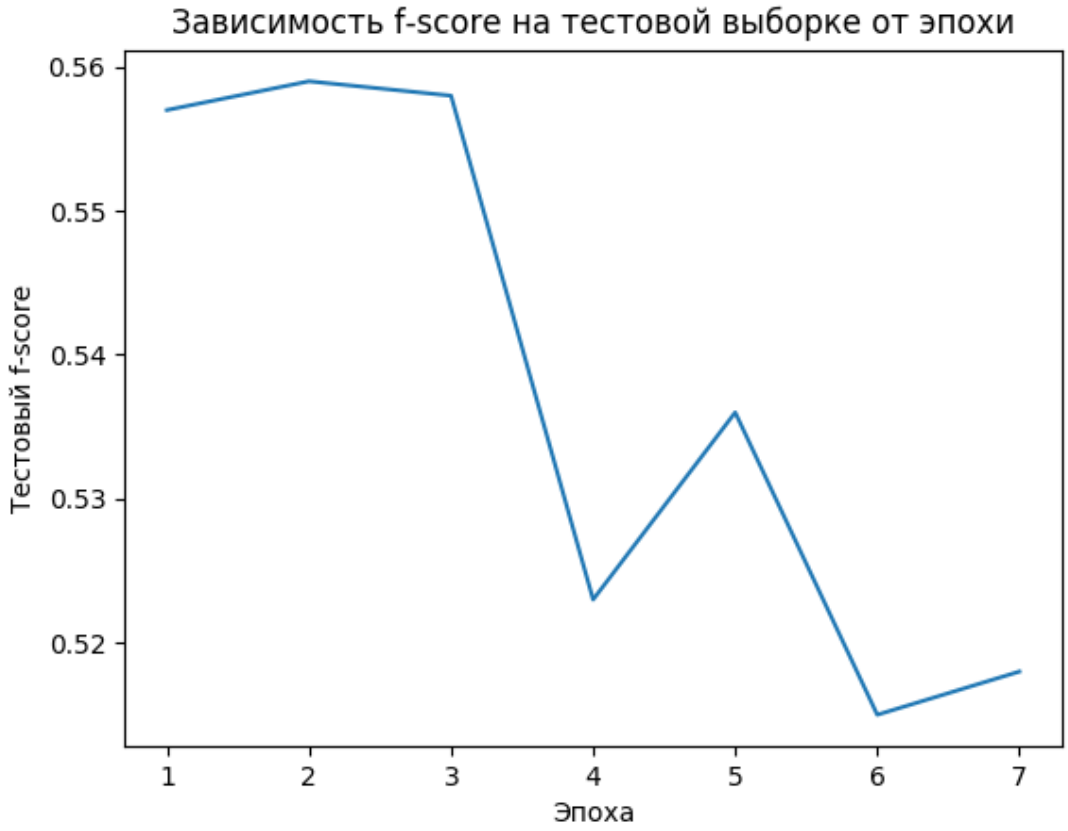

Рисунок 3.3 – Изменение тестового f-score для русского набора данных

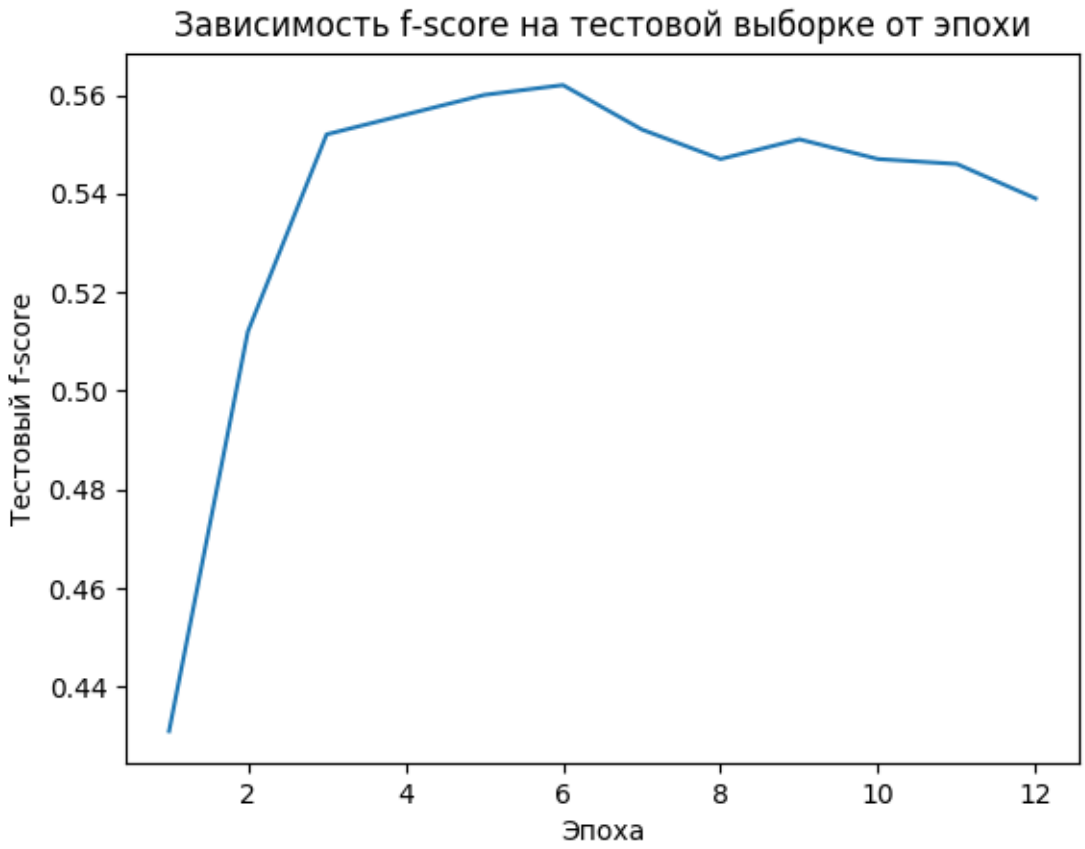

Рисунок 3.4 – Изменение тестового f-score для английского набора данных

Из графиков можно сделать вывод, что после нескольких эпох происходит переобучение, так как оценки на тренировочных выборках продолжают расти, а на тестовых начинают уменьшаться.

3.3 Применение обученных классификаторов

Полученные модели были применены при реализации веб-сервиса доступного для всеобщего пользования в сети Интернет [29].

Использовалась библиотека Flask [30] и её расширение Flask-Cors [31], а также языки HTML, CSS и JavaScript.

Интерфейс сервиса для белорусскоязычных пользователей представлен на рисунке 3.5.

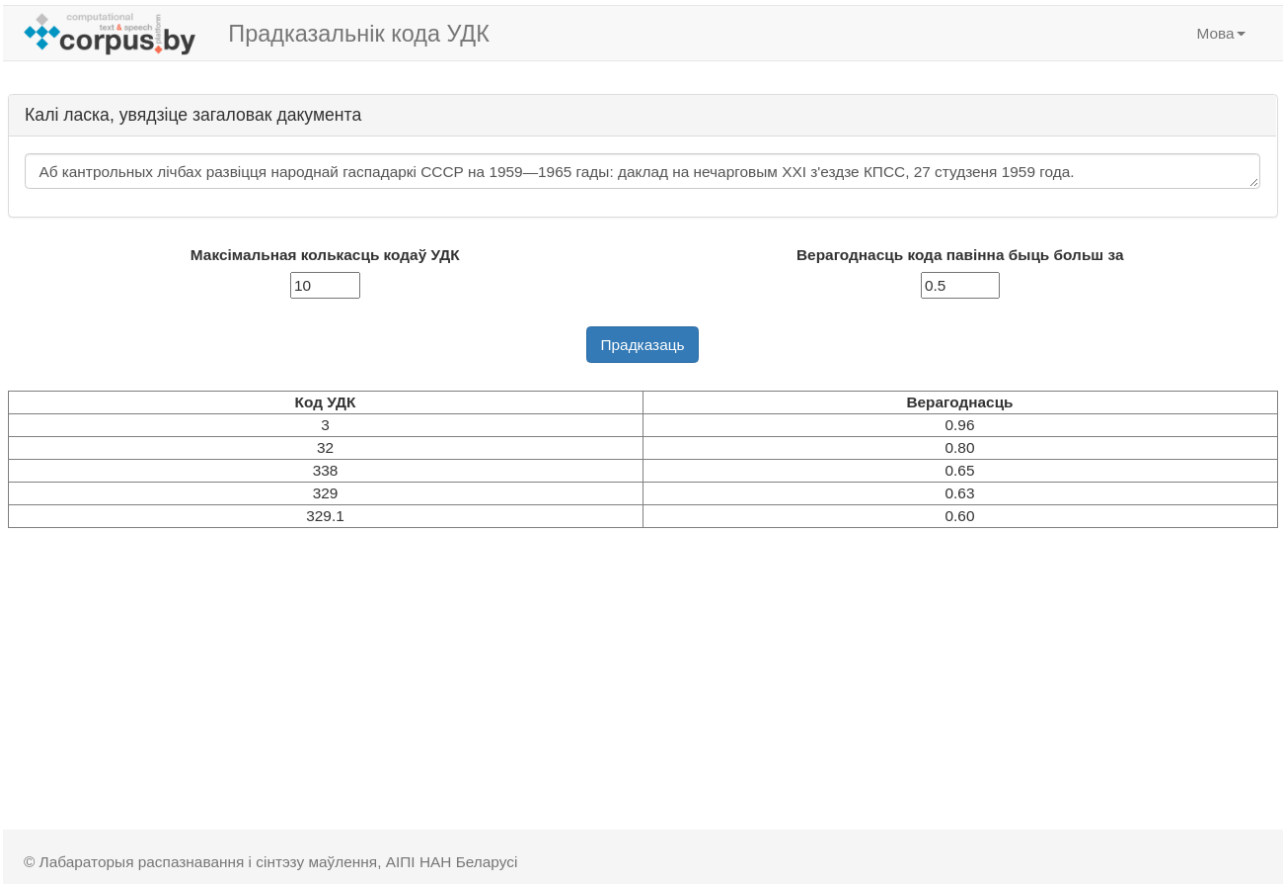

Рисунок 3.5 – Интерфейс реализованного веб-сервиса на белорусском языке

Пользователь должен ввести заголовок книги, нажать на кнопку предсказания и получить один или несколько возможных кодов УДК для этого документа и указанных параметров.

Сервис называется «Прадказальнік кода УДК», «Предсказатель УДК» или «UDC Predictor» для пользователей, владеющих, соответственно, белорусским, русским или английским языком. Не только название, но и весь интерфейс сервиса локализован для этих 3 языков.

3.4 Выводы

В этой главе был описан использованный алгоритм машинного обучения для автоматической классификации заголовков книг на белорусском, русском и английском языках по системе УДК. В частности, была рассмотрена архитектура использованной нейронной сети, использованные параметры и характеристики входных данных.

Для полученных в результате обучения моделей продемонстрированы оценки точности предсказания. В частности, представлены такие популярные метрики как «precision», «recall» и «f-score» как на трениворочной, так и на тестовой выборке.

Готовые модели были использованы при создании общедоступного веб-сервиса для автоматической классификации документов по системе УДК.

Простой интерфейс не требует специального обучения для использования сервиса.

В завершающей главе будет сделано краткое резюме полученных результатов и рассмотрены дальнейшие планы и возможные направления исследования.

ЗАКЛЮЧЕНИЕ

В результате проделанной работы был разработан программный комплекс для автоматического сбора набора данных кодов УДК для документов на белорусском, русском и английском языках.

С помощью него были собраны собственные наборы данных заголовков книг на указанных языках с проставленными для них кодами УДК.

На собранных наборах данных были обучены модели для автоматического предсказания кодов УДК по заголовкам книг.

Эти модели были использованы в разработке и публикации в сети Интернет общедоступного веб-сервиса для предсказания кодов УДК.

В ходе дальнейшей работы планируется улучшение текущих моделей предсказания (увеличение времени обучения, добавление прочей метаинформации в обучающую выборку, использование других архитектур сети), обучение отдельных моделей для определителей места и времени, интеграция в реализованный веб-сервис возможности получения текстового описания кода УДК.

С помощью предоставленных в приложениях скриптов исследование может быть продолжено и расширено путём сбора иных наборов данных, в том числе на других языках, а также другой их предобработки или использования других алгоритмов машинного обучения.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Corpus.by. Computational platform for electronic text & speech processing [Electronic resource] – 2012-2021. – Mode of access: https://corpus.by/. – Date of access: 16.03.2021.
2. Wikipedia.org. ВикипедиЯ – свободная энциклопедия [Электронный ресурс] – 2001-2021. – Способ доступа: https://ru.wikipedia.org/wiki/Универсальная_десятичная_классификация. – Дата доступа: 16.03.2021.
3. Універсальная дзесятковая класіфікацыя : звыш 10 000 асноўных і дапаможных класаў / Аб'яднаны інстытут праблем інфарматыкі Нацыянальнай акадэміі навук Беларусі, Нацыянальная бібліятэка Беларусі; [рэдакцыйная калегія: Ю. С. Гецэвіч, С. А. Пугачова, Г. Р. Станіславенка і інш. ; укладальнікі алфавітна-прадметнага паказальніка: С. І. Лысы, Г. Р. Станіславенка, Ю. С. Гецэвіч]. – Мінск, 2016. – 370 с..
4. Udcc.org. UDC Consortium [Electronic resource] – 1992-2021. – Mode of access: https://udcc.org/. – Date of access: 16.03.2021.
5. Automatic Universal Decimal Classification based on frequency advanced method / Fiodor Tretyakov, Liya Serebryanaya // Central European Researchers Journal – Vol.1 Issue 1 – 2015.
6. E-catalog.nlb.by. Электронный каталог Национальной библиотеки Республики Беларусь [Электронный ресурс] – 2006-2021. – Способ доступа: https://e-catalog.nlb.by/. – Date of access: 16.03.2021.
7. Python requests. HTTP for Humans [Electronic resource] – 2011-2021. – Mode of access: https://github.com/psf/requests. – Date of access: 16.03.2021.
8. Beautiful Soup. A Python library for pulling data out of HTML and XML files [Electronic resource] – 2004-2021. – Mode of access: https://pypi.org/project/beautifulsoup4/. – Date of access: 16.03.2021.
9. BibTeX.org [Electronic resource] – 2006-2021. – Mode of access: http://www.bibtex.org/. – Date of access: 16.03.2021.
10. Matplotlib: Visualization with Python [Electronic resource] – 2012-2021. – Mode of access: https://matplotlib.org/. – Date of access: 16.03.2021.
11. Corpus.by. Computational platform for electronic text & speech processing [Electronic resource] – 2012-2021. – Mode of access: https://corpus.by/UdcDecoder/. – Date of access: 16.03.2021.
12. Wikipedia.org. WikipediA – The Free Encyclopedia [Electronic resource] – 2001-2021. – Mode of access: https://en.wikipedia.org/wiki/Multiclass_classification. – Date of access: 16.03.2021.
13. Tatoeba. A collection of sentences and translations. [Electronic resource] – 2006-2021. – Mode of access: https://tatoeba.org/. – Date of access: 16.03.2021.

14. SETimes. A Parallel Corpus of English and South-East European Languages [Electronic resource] – 2012-2021. – Mode of access: http://nlp.ffzg.hr/resources/corpora/setimes/. – Date of access: 16.03.2021.
15. fastText. Library for efficient text classification and representation learning [Electronic resource] – 2016-2021. – Mode of access: https://fasttext.cc/docs/en/language-identification.html. – Date of access: 16.03.2021.
16. spaCy. Industrial-Strength Natural Language Processing [Electronic resource] – 2016-2021. – Mode of access: https://spacy.io/. – Date of access: 16.03.2021.
17. datrie. Fast, efficiently stored Trie for Python [Electronic resource] – 2012-2021. – Mode of access: https://github.com/pytries/datrie. – Date of access: 16.03.2021.
18. regex. Alternative regular expression module, to replace re [Electronic resource] – 2013-2021. – Mode of access: https://pypi.org/project/regex/. – Date of access: 16.03.2021.
19. Wikipedia.org. ВикипедиЯ – свободная энциклопедия [Электронный ресурс] – 2001-2021. – Способ доступа: https://ru.wikipedia.org/wiki/Регулярные_выражения. – Дата доступа: 16.03.2021.
20. Wikipedia.org. WikipediA – The Free Encyclopedia [Electronic resource] – 2001-2021. – Mode of access: https://en.wikipedia.org/wiki/Multiclass_classification. – Date of access: 16.03.2021.
21. Wikipedia.org. ВикипедиЯ – свободная энциклопедия [Электронный ресурс] – 2001-2021. – Способ доступа: https://ru.wikipedia.org/wiki/Таксономия. – Дата доступа: 16.03.2021.
22. Wikipedia.org. ВикипедиЯ – свободная энциклопедия [Электронный ресурс] – 2001-2021. – Способ доступа: https://ru.wikipedia.org/wiki/Префиксное_дерево. – Дата доступа: 16.03.2021.
23. spaCy. Industrial-Strength Natural Language Processing [Electronic resource] – 2016-2021. – Mode of access: https://github.com/explosion/spacy-models/releases//tag/ru_core_news_md-3.0.0. – Date of access: 16.03.2021.
24. spaCy. Industrial-Strength Natural Language Processing [Electronic resource] – 2016-2021. – Mode of access: https://github.com/explosion/spacy-models/releases//tag/en_core_web_md-3.0.0. – Date of access: 16.03.2021.
25. NeuralNLP-NeuralClassifier. An Open-source Neural Hierarchical Multi-label Text Classification Toolkit [Electronic resource] – 2019-2021. – Mode of access: https://github.com/Tencent/NeuralNLP-NeuralClassifier. – Date of access: 16.03.2021.
26. Recurrent Neural Network for Text Classification with Multi-Task Learning / Pengfei Liu, Xipeng Qiu, Xuanjing Huang // Shanghai Key Laboratory of Intelligent Information Processing, Fudan University – China – 2016.
27. BCEWithLogitsLoss. PyTorch loss function [Electronic resource] – 2019-2021. – Mode of access: https://pytorch.org/docs/stable/generated/torch.nn.BCEWithLogitsLoss.html. – Date of access: 16.03.2021.

28. Adam: A Method for Stochastic Optimization / Diederik P. Kingma, Jimmy Ba // 3rd International Conference for Learning Representations – San Diego – 2015.
29. Corpus.by. Computational platform for electronic text & speech processing [Electronic resource] – 2012-2021. – Mode of access: https://corpus.by/UDCPredictor/. – Date of access: 16.03.2021.
30. Flask. Lightweight WSGI web application framework [Electronic resource] – 2010-2021. – Mode of access: https://pypi.org/project/Flask/. – Date of access: 16.03.2019.
31. Flask-Cors. Flask extension adding a decorator for CORS support [Electronic resource] – 2013-2019. – Mode of access: https://pypi.org/project/Flask-Cors/. – Date of access: 16.03.2021.

ПРИЛОЖЕНИЕ А

ИСХОДНЫЙ КОД СКРИПТА ДЛЯ СБОРА ДАННЫХ

```
import argparse
import datetime
import functools
import logging
import pathlib
import pickle

import biblib.algo
import biblib.bib
import biblib.messages
import bs4
import compress_pickle
import requests
import tqdm

import argparse_utils
import crawling_utils


def create_cmd_argparser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(description='The National Library of Belarus E-
Catalogue crawler.',
                                     usage='%(prog)s [-h] data_file log_file [start_page] [end_page]
[--option]...',
                                     add_help=False)

    files_group = parser.add_argument_group('Files', 'Arguments related to files.')
    files_group.add_argument('data_file',
                             type=pathlib.Path,
                             help=('the file path for the collected data (type: FileType(\'wb\'), '
                                   'the file is opened only after the --prev_data_file reading and
closing '
                                   'to support read and write to the same file)'))
    files_group.add_argument('log_file',
                             type=pathlib.Path,
                             help='the file path for logging (type: FileType(\'a\'))')
    files_group.add_argument('--prev_data_file',
                             type=pathlib.Path,
```

```
                    help=('the file path for the previously collected data '
                        'to concatenate with the current data (type: FileType(\'rb\'), '
                        'the file must be in pickle format, optionally compressed by '
                        'gzip, bz2, lzma, zip or lz4 compression standard '
                        '(the compression standard is inferred from the path\'s
extension))'),
                    metavar='FILE')
    files_group.add_argument('--compression', '-c',
                    type=str, default='lz4',
                    choices=['gzip', 'bz2', 'lzma', 'zip', 'lz4', 'none'],
                    help=('the compression standard for the collected data '
                        '(type: %(type)s, default: %(default)s)'))

    pages_group = parser.add_argument_group('Pages', 'Arguments related to the results
pages.')
    pages_group.add_argument('start_page', nargs='?',
                    type=functools.partial(argparse_utils.ranged_int, min_val=1),
default=1,
                    help='the first page to process (type: positive int, default:
%(default)s)')
    pages_group.add_argument('end_page', nargs='?',
                    type=functools.partial(argparse_utils.ranged_int, min_val=1),
                    help='the last page to process (type: positive int, default: all available
pages)')
    pages_group.add_argument('--limit', '-l',
                    type=int, default=100, choices=[20, 50, 100],
                    help='the number of documents per page (type: %(type)s, default:
%(default)s)')

    query_group = parser.add_argument_group('Query', 'Arguments related to the query
string.')
    query_group.add_argument('--lookfor', '-q',
                    type=str, default='',
                    help=('the query string to search for in the document field '
                        '(type: %(type)s, default: any query)'),
                    metavar='QUERY')
    query_group.add_argument('--type', '-t',
                    type=str, default='AllFields', choices=['AllFields', 'Title', 'Author',
'Subject', 'ISN'],
                    help=('the document field in which to search for the query string '
                        '(type: %(type)s, default: %(default)s)'))
    query_group.add_argument('--sort', '-s',
```

```
                    type=str, default='relevance',
                    choices=['relevance', 'year', 'year+asc', 'author', 'title'],
                    help=('the documents sort order by the query string '
                        '(type: %(type)s, default: %(default)s)'))

    filters_group = parser.add_argument_group('Filters', 'Arguments related to the search
filters.')
    filters_group.add_argument('--national_doc', action='store_true',
                        help=('search only for The Belarusian National Document'
                            '(type: bool, default: False)'))
    filters_group.add_argument('--format',
                        type=str, default='Book',
                        choices=['Book', 'AuthorDissertation', 'Article',
'MusicalDocumentation',
                                 'Dissertation', 'Image', 'Serial_Continue', 'Serial',
                                 'Rare', 'AudioVideo', 'Cartographic', 'InternationalOrg',
                                 'Manuscripts', 'Electronic', 'Standardization', 'Reviews',
'Scientifically'],
                        help='search only for this document format/type (type: %(type)s,
default: %(default)s)')
    filters_group.add_argument('--author_sort', '--author',
                        type=str,
                        help='search only for this main author documents (type: %(type)s,
default: any author)',
                        metavar='AUTHOR')
    filters_group.add_argument('--topic_facet', '--topic', nargs='+',
                        type=str,
                        help='search only for documents with these topics (type: %(type)s,
default: any topic)',
                        metavar='TOPIC')
    filters_group.add_argument('--genre_facet', '--genre', nargs='+',
                        type=str,
                        help='search only for documents with these genres (type: %(type)s,
default: any genre)',
                        metavar='GENRE')
    filters_group.add_argument('--series_facet', '--series', nargs='+',
                        type=str,
                        help=('search only for documents that are parts of these series'
                            ' (type: %(type)s, default: any series)'),
                        metavar='SERIES')
    filters_group.add_argument('--publishPlace', '--place', nargs='+',
                        type=str,
```

```
                    help=('search only for documents published in these places'
                        ' (type: %(type)s, default: any place)'),
                    metavar='PLACE')
    filters_group.add_argument('--language', '--lang', nargs='+',
                    type=str,
                    help=('search only for documents containing all of these languages'
                        ' (type: %(type)s, default: any language)'),
                    metavar='LANG')
    filters_group.add_argument('--publishDatefrom', '--from',
                    type=functools.partial(argparse_utils.ranged_int,
                                           min_val=1400,
max_val=datetime.datetime.now().year + 1),
                    help=('search only for documents published from this year'
                        f' (type: int in [1400; {datetime.datetime.now().year + 1}],'
                        ' default: 1400)'),
                    metavar='YEAR')
    filters_group.add_argument('--publishDateto', '--to',
                    type=functools.partial(argparse_utils.ranged_int,
                                           min_val=1400,
max_val=datetime.datetime.now().year + 1),
                    help=('search only for documents published by this year'
                        f' (type: int in [1400; {datetime.datetime.now().year + 1}],'
                        f' default: {datetime.datetime.now().year + 1})'),
                    metavar='YEAR')

    other_group = parser.add_argument_group('Other', 'Other arguments.')
    other_group.add_argument('--help', '-h', action='help',
                        help='show this help message and exit')
    other_group.add_argument('--timeout',
                        type=functools.partial(argparse_utils.ranged_float, min_val=0.),
default=30.,
                        help='the number of seconds to wait for the server to send data before
giving up'
                            ' (type: non-negative float, default: %(default)s)',
                        metavar='NUM')
    other_group.add_argument('--retries', '-r',
                        type=functools.partial(argparse_utils.ranged_int, min_val=0),
default=5,
                        help='the number of retries after a failed request'
                            ' (type: non-negative int, default: %(default)s)',
                        metavar='NUM')
```

```
    return parser


def form_search_params(cmd_kwargs: dict[str]) -> dict[str]:
    search_params: dict[str] = dict()

    for key in ('lookfor', 'type', 'sort', 'limit'):
        search_params[key] = cmd_kwargs[key]
    if cmd_kwargs['national_doc']:
        search_params.setdefault('filter[]', []).append('national_doc:1')
    if cmd_kwargs['format'] is not None:
        search_params.setdefault('filter[]',
[]).append(f'"format:"Format_{cmd_kwargs['format']}"')
    if cmd_kwargs['author_sort'] is not None:
        search_params.setdefault('filter[]',
[]).append(f'"author_sort:"{cmd_kwargs['author_sort']}"')
    for key in ('topic_facet', 'genre_facet', 'series_facet', 'publishPlace', 'language'):
        search_params.setdefault('filter[]', []).extend(f'{key}:{val}' for val in
cmd_kwargs[key] or [])
    if cmd_kwargs['publishDatefrom'] is not None or cmd_kwargs['publishDateto'] is not
None:
        search_params['daterange[]'] = 'publishDate'
        search_params['publishDatefrom'] = cmd_kwargs['publishDatefrom'] or ''
        search_params['publishDateto'] = cmd_kwargs['publishDateto'] or ''

    return search_params


def form_export_params(result_div_tags: bs4.element.ResultSet) -> dict[str]:
    export_params: dict[str] = dict()

    export_params['f'] = 'BibTex'
    export_params['i[]'] = [f'"{div_tag.find('input', {'class': 'hiddenSource'})['value']}"'
                            f'"|"'
                            f'"{div_tag.find('input', {'class': 'hiddenId'})['value']}"'
                            for div_tag in result_div_tags]

    return export_params


def crawl(base_url: str, search_query: str, export_query: str, cmd_kwargs: dict[str]) ->
dict[str, dict[str]]:
```

```
    logging.info(f'crawling started: {base_url}')

    search_query_params = form_search_params(cmd_kwargs)

    logging.info(f'search parameters: {search_query_params}')

    start_page = cmd_kwargs['start_page']
    end_page = cmd_kwargs['end_page']
    last_processed_page = None

    session = crawling_utils.Session(base_url=base_url, timeout=cmd_kwargs['timeout'],
retries=cmd_kwargs['retries'])

    if end_page is None:
        response = session.get(search_query, params=search_query_params)
        parsed_html = bs4.BeautifulSoup(response.text, 'html.parser')

        end_page = int(parsed_html.find('ul', {'class': 'pagination'}).find_all('li')[-
1].get_text()[1:-1])

    logging.info(f'pages to process: from {start_page} to {end_page}')

    collected_data: dict[str, dict[str]] = dict()

    # noinspection PyBroadException
    try:
        for cur_page in tqdm.tqdm(range(start_page, end_page + 1), initial=start_page,
total=end_page,
                                  unit='page', dynamic_ncols=True):
            try:
                search_query_params['page'] = cur_page
                search_response = session.get(search_query, params=search_query_params)

                parsed_html = bs4.BeautifulSoup(search_response.text, 'html.parser')
                result_div_tags = parsed_html.findAll('div', {'class': 'result'})

                export_query_params = form_export_params(result_div_tags)
                export_response = session.get(export_query, params=export_query_params)

                parsed_bibtex = biblib.bib.Parser().parse(export_response.text)

                for entry in parsed_bibtex.get_entries().values():
```

```
                collected_data[entry.key] = dict(entry)

            logging.info(f'data collected: {cur_page}th page')
        except biblib.messages.InputError:
            logging.warning(f'bibtex parsing failed: skipping {cur_page}th page')

        last_processed_page = cur_page
    except requests.RequestException:
        logging.exception(f'request failed {session.retries + 1} times: finishing')
    except KeyboardInterrupt:
        logging.exception(f'SIGINT caught: finishing')
    except BaseException:
        logging.exception(f'unknown exception caught: finishing')

    logging.info(f'pages processed: from {start_page} to {last_processed_page}'
                 if last_processed_page is not None else 'pages processed: 0')
    logging.info(f'crawling finished: {base_url}')

    return collected_data


def main(cmd_kwargs: dict[str]) -> None:
    logging.basicConfig(filename=cmd_kwargs['log_file'],
                        format='%(asctime)s | %(levelname)-8s | %(message)s',
                        level=logging.INFO)

    old_data = (compress_pickle.load(cmd_kwargs['prev_data_file'],
set_default_extension=False)
                if cmd_kwargs['prev_data_file'] else dict())

    with open(cmd_kwargs['data_file'], 'wb') as data_file:
        new_data = crawl('https://e-catalog.nlb.by', '/Search/Results', '/Cart/doExport',
cmd_kwargs)

        compress_pickle.dump(old_data | new_data,
                             data_file,
                             compression=cmd_kwargs['compression'] if
cmd_kwargs['compression'] != 'none' else None,
                             pickler_kwargs={'protocol': pickle.HIGHEST_PROTOCOL})

if __name__ == '__main__':
    main(vars(create_cmd_argparser().parse_args()))
```

ПРИЛОЖЕНИЕ Б

ИСХОДНЫЙ КОД СКРИПТА ДЛЯ АНАЛИЗА ДАННЫХ

```
import argparse
import collections
import functools
import logging
import pathlib
import re
import typing

import compress_pickle
import fasttext
import matplotlib.pyplot as plt
import requests
import tqdm

import argparse_utils
import crawling_utils


def create_cmd_argparser() -> argparse.ArgumentParser:
    parser = argparse.ArgumentParser(description='The National Library of Belarus E-
Catalogue data analyzer.',
                                     usage='%(prog)s [-h] data_file log_file bar_plots_dir [--
option]...',
                                     add_help=False)

    files_group = parser.add_argument_group('Files', 'Arguments related to files.')
    files_group.add_argument('data_file',
                             type=pathlib.Path,
                             help='the file path for the collected data (type: FileType(\'rb\'))')
    files_group.add_argument('log_file',
                             type=pathlib.Path,
                             help='the file path for logging (type: FileType(\'a\'))')
    files_group.add_argument('bar_plots_dir',
                             type=pathlib.Path,
                             help='the dir path for bar plots (type: %(type)s)')

    other_group = parser.add_argument_group('Other', 'Other arguments.')
    other_group.add_argument('--help', '-h', action='help',
```

```
                    help='show this help message and exit')
    other_group.add_argument('--timeout',
                    type=functools.partial(argparse_utils.ranged_float, min_val=0.),
default=10.,
                    help='the number of seconds to wait for the server to send data before
giving up'
                        ' (type: non-negative float, default: %(default)s)',
                    metavar='NUM')
    other_group.add_argument('--retries', '-r',
                    type=functools.partial(argparse_utils.ranged_int, min_val=0),
default=1,
                    help='the number of retries after a failed request'
                        ' (type: non-negative int, default: %(default)s)',
                    metavar='NUM')

    return parser


def produce_bar_chart(code_counter: collections.Counter, n: int, path: pathlib.Path,
                      decode_codes: bool = False, cmd_kwargs: typing.Optional[dict[str]] =
None) -> None:
    codes, frequencies = zip(*code_counter.most_common(n))
    captions = list(map(str, codes))

    if decode_codes:
        session = crawling_utils.Session(base_url='https://corpus.by/UdcDecoder/',
                                         timeout=cmd_kwargs['timeout'],
retries=cmd_kwargs['retries'])

        for index, code in enumerate(captions):
            if set(code) != {'0'}:
                decoded_code = None

                try:
                    decoded_code = session.post('api.php', data={'text': code}).json()['english']
                except requests.HTTPError:
                    logging.warning(f'request failed {session.retries + 1} times: skipping
{code}')

                if decoded_code and len(decoded_code) <= 30:
                    captions[index] = decoded_code
            else:
```

```
            captions[index] = 'Science and knowledge'

    bars = plt.bar(captions, frequencies)
    plt.xticks(rotation=290, fontsize=7)

    for bar in bars:
        x, height = bar.get_x(), bar.get_height()
        plt.text(x, height + .005, height, fontsize=6)

    plt.tight_layout()
    plt.savefig(path)
    plt.clf()


def main(cmd_kwargs: dict[str]) -> None:
    logging.basicConfig(filename=cmd_kwargs['log_file'],
                        format='%(asctime)s | %(levelname)-8s | %(message)s',
                        level=logging.INFO)

    logging.info(f'"analyzing started: {cmd_kwargs['data_file']}"')

    data = compress_pickle.load(cmd_kwargs['data_file'], set_default_extension=False)
    meta_data = [meta for meta in data.values()]

    meta_keys_counter = collections.Counter()
    lang_in_language_key_counter = collections.Counter()

    for meta in meta_data:
        for key in meta.keys():
            meta_keys_counter[key] += 1

        if 'language' in meta:
            for lang in meta['language'].split(' and '):
                lang_in_language_key_counter[lang] += 1

    print()
    print(f'total records = {len(data)}')
    print(f'meta keys = {meta_keys_counter}')
    print(f'language keys = {lang_in_language_key_counter}')
    print()

    lang_in_title_counter = collections.Counter()
```

```
    with_slash_in_title = 0
    with_slash_surrounded_by_spaces_in_title = 0
    with_hyphen_in_title = 0
    with_hyphen_surrounded_by_spaces_in_title = 0
    with_dash_in_title = 0
    with_dash_surrounded_by_spaces_in_title = 0
    without_slash_hyphen_or_dash_in_title = 0

    model = fasttext.load_model('models/lid.176.bin')

    for meta in tqdm.tqdm(meta_data):
        lang = model.predict(meta['title'])[0][0].split('__label__')[1]

        lang_in_title_counter[lang] += 1

        if '/' in meta['title']:
            with_slash_in_title += 1
        if ' / ' in meta['title']:
            with_slash_surrounded_by_spaces_in_title += 1
        if '-' in meta['title']:
            with_hyphen_in_title += 1
        if ' - ' in meta['title']:
            with_hyphen_surrounded_by_spaces_in_title += 1
        if '—' in meta['title']:
            with_dash_in_title += 1
        if ' — ' in meta['title']:
            with_dash_surrounded_by_spaces_in_title += 1
        if '/' not in meta['title'] and '-' not in meta['title'] and '—' not in meta['title']:
            without_slash_hyphen_or_dash_in_title += 1

    print()
    print(f'lang in title = {lang_in_title_counter}')
    print(f'contain "/" in title = {with_slash_in_title}')
    print(f'contain " / " in title = {with_slash_surrounded_by_spaces_in_title}')
    print(f'contain "-" in title = {with_hyphen_in_title}')
    print(f'contain " - " in title = {with_hyphen_surrounded_by_spaces_in_title}')
    print(f'contain "—" in title = {with_dash_in_title}')
    print(f'contain " — " in title = {with_dash_surrounded_by_spaces_in_title}')
    print(f'doesn\'t contain "/", "-", "—" in title =
{without_slash_hyphen_or_dash_in_title}')
    print()
```

```
    main_book_classes = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
    main_teacode_classes = ['00', '1', '2', '30', '31', '32', '33', '34', '35', '36', '37', '39', '50',
'51', '52',
                            '53', '54', '55', '56', '57', '58', '59', '60', '61', '62', '63', '64', '65', '66',
                            '67', '68', '69', '7', '8', '9']
    plain_symbols = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9', '.'}

    codes = [meta['udk'] or meta['UDK'] for meta in meta_data if 'udk' in meta or 'UDK'
in meta]
    unique_codes = set(codes)
    unique_symbols = set(symbol for code in unique_codes for symbol in code)
    non_alphanumeric_symbols = set(symbol for symbol in unique_symbols if not
symbol.isalnum())
    with_dot = 0
    with_plus = 0
    with_and = 0
    with_and_surrounded_by_whitespace = 0
    with_slash = 0
    with_colon = 0
    with_double_colon = 0
    with_asterisk = 0
    with_lang_det = 0
    with_form_det = 0
    with_place_det = 0
    with_time_det = 0
    first_symbol_counter = collections.Counter()
    first_two_symbols_counter = collections.Counter()
    plain_digits_and_dots_counter = collections.Counter()
    main_book_counter = collections.Counter()
    main_teacode_counter = collections.Counter()
    non_alphanumeric_symbols_counter = collections.Counter()

    for code in tqdm.tqdm(codes):
        if re.search(r'(?<!\()=.*', code):
            with_lang_det += 1
        if re.search(r'\(0.*\)', code):
            with_form_det += 1
        if re.search(r'\([1-9].*\)', code):
            with_place_det += 1
        if re.search(r'".*"', code):
            with_time_det += 1
```

```
    if '.' in code:
        with_dot += 1
    if '+' in code:
        with_plus += 1
    if 'and' in code:
        with_and += 1
    if re.search(r'\s+and\s+', code):
        with_and_surrounded_by_whitespace += 1
    if '/' in code:
        with_slash += 1
    if ':' in code:
        with_colon += 1
    if '::' in code:
        with_double_colon += 1
    if '*' in code:
        with_asterisk += 1

    first_symbol_counter[code[0]] += 1
    first_two_symbols_counter[code[0:2]] += 1

    if plain_symbols.issuperset(code):
        plain_digits_and_dots_counter[code] += 1

    for main_book_class in main_book_classes:
        if code[0] == main_book_class:
            main_book_counter[main_book_class] += 1

    for main_teacode_class in main_teacode_classes:
        if code.startswith(main_teacode_class):
            main_teacode_counter[main_teacode_class] += 1

    for symbol in non_alphanumeric_symbols:
        if symbol in code:
            non_alphanumeric_symbols_counter[symbol] += 1

print()
print(f'UDC code specified = {len(codes)}')
print(f'unique UDC code = {len(unique_codes)}')
print(f'code symbols = {unique_symbols}')
print(f'non-alphanumeric symbols = {non_alphanumeric_symbols}')
print(f'plain digits and dots = {sum(plain_digits_and_dots_counter.values())}')
print(f'unique plain digits and dots = {len(plain_digits_and_dots_counter)}')
```

```
    print(f'contain "." = {with_dot}')
    print(f'contain "+" = {with_plus}')
    print(f'contain "and" = {with_and}')
    print(f'contain "and" surrounded by whitespace =
{with_and_surrounded_by_whitespace}')
    print(f'contain "/" = {with_slash}')
    print(f'contain ":" = {with_colon}')
    print(f'contain "::" = {with_double_colon}')
    print(f'contain "*" = {with_asterisk}')
    print(f'contain lang det = {with_lang_det}')
    print(f'contain form det = {with_form_det}')
    print(f'contain place det = {with_place_det}')
    print(f'contain time det = {with_time_det}')
    print(f'first symbol = {first_symbol_counter}')
    print(f'first two symbols = {first_two_symbols_counter}')
    print(f'main book classes = {main_book_counter}')
    print(f'teacode main classes = {main_teacode_counter}')
    print(f'non-alphanumeric symbols frequency =
{non_alphanumeric_symbols_counter}')
    print()

    counters = {name: value for name, value in locals().items() if
name.endswith('_counter')}
    for name, counter in tqdm.tqdm(counters.items()):
        produce_bar_chart(counter, 10, cmd_kwargs['bar_plots_dir'] / f'{name}.png', True,
cmd_kwargs)

    logging.info(f'"analyzing finished: {cmd_kwargs['data_file']}"')


if __name__ == '__main__':
    main(vars(create_cmd_argparser().parse_args()))
```

ПРИЛОЖЕНИЕ В

ИСХОДНЫЙ КОД СКРИПТА ДЛЯ ИЗВЛЕЧЕНИЯ ДАННЫХ

```
import argparse
import pathlib
import pickle

import compress_pickle
import fasttext
import tqdm


def create_cmd_argparser() -> argparse.ArgumentParser:
   parser = argparse.ArgumentParser(description='The National Library of Belarus E-
Catalogue data extractor.',
                                    usage='%(prog)s [-h] data_file... [--option]...',
                                    add_help=False)

   files_group = parser.add_argument_group('Files', 'Arguments related to files.')
   files_group.add_argument('data_files', nargs='+',
                            type=pathlib.Path,
                            help='file paths for the collected data (type: FileType(\'rb\'))')
   files_group.add_argument('--extracted_data_file', '-o',
                            type=pathlib.Path, default='data.pickle.lz4',
                            help='the file path for the extracted data (type: %(type)s, default:
%(default)s)',
                            metavar='DIR')
   files_group.add_argument('--compression', '-c',
                            type=str, default='lz4',
                            choices=['gzip', 'bz2', 'lzma', 'zip', 'lz4', 'none'],
                            help=('the compression standard for the extracted data '
                                  '(type: %(type)s, default: %(default)s)'))

   data_group = parser.add_argument_group('Data', 'Arguments related to the data
extraction.')
   data_group.add_argument('--keys', '-k', nargs='+',
                           type=str, default=['title', 'udk'],
                           help='extract data with these keys, "title" key is required if --language
is not \'any\'"
                                ' (type: %(type)s, default: %(default)s)',
                           metavar='KEY')
```

```
    data_group.add_argument('--language', '-l',
                            type=str, default='any', choices=['be', 'ru', 'en', 'any'],
                            help='extract data with this language in title (type: %(type)s, default:
%(default)s)',
                            metavar='LANG')

    other_group = parser.add_argument_group('Other', 'Other arguments.')
    other_group.add_argument('--help', '-h', action='help',
                             help='show this help message and exit')

    return parser


def main(cmd_kwargs: dict[str]) -> None:
    lang_detection_model = fasttext.load_model('models/lid.176.bin')

    extracted_data: list[dict[str]] = list()

    for data_file in cmd_kwargs['data_files']:
        data = compress_pickle.load(data_file, set_default_extension=False)

        for meta in tqdm.tqdm(data.values()):
            if all(key in meta for key in cmd_kwargs['keys']):
                if cmd_kwargs['language'] != 'any':
                    lang = lang_detection_model.predict(meta['title'])[0][0].split('__label__')[1]
                else:
                    lang = 'any'

                if lang == cmd_kwargs['language']:
                    extracted_data.append({key: meta[key] for key in cmd_kwargs['keys']})

    compress_pickle.dump(extracted_data,
                         cmd_kwargs['extracted_data_file'],
                         compression=cmd_kwargs['compression'] if cmd_kwargs['compression']
!= 'none' else None,
                         pickler_kwargs={'protocol': pickle.HIGHEST_PROTOCOL})


if __name__ == '__main__':
    main(vars(create_cmd_argparser().parse_args()))
```

ПРИЛОЖЕНИЕ Г

ИСХОДНЫЙ КОД СКРИПТА ДЛЯ ОБРАБОТКИ ДАННЫХ

```
import argparse
import functools
import json
import pathlib
import random
import re
import string
import typing
import heapq
import operator
import bisect

import en_core_web_md
import ru_core_news_md
import spacy
import compress_pickle
import datrie
import regex
import tqdm

import argparse_utils


def create_cmd_argparser() -> argparse.ArgumentParser:
   parser = argparse.ArgumentParser(description='The National Library of Belarus E-
Catalogue data preprocessor.',
                                    usage='%(prog)s [-h] data_file... [--option]...',
                                    add_help=False)

   files_group = parser.add_argument_group('Files', 'Arguments related to files.')
   files_group.add_argument('extracted_data_file',
                            type=pathlib.Path,
                            help='the file path for the extracted data (type: FileType(\'rb\'))')
   files_group.add_argument('preprocessed_data_dir',
                            type=pathlib.Path,
                            help='the dir path for the preprocessed data (type: %(type)s)')
```

```
    data_group = parser.add_argument_group('Data', 'Arguments related to the data
preprocessing.')
    data_group.add_argument('--data_key', '-X',
                            type=str, default='title',
                            help='the extracted data key for training (data/X) (type: %(type)s,
default: %(default)s)',
                            metavar='KEY')
    data_group.add_argument('--label_key', '-Y',
                            type=str, default='udk',
                            help='the extracted data key for training (label/Y) (type: %(type)s,
default: %(default)s)',
                            metavar='KEY')
    data_group.add_argument('--top', '-t',
                            type=functools.partial(argparse_utils.ranged_int, min_val=1),
default=100,
                            help='extract only the top NUM labels (type: int in [0; +inf], default:
%(default)s)',
                            metavar='NUM')
    data_group.add_argument('--drop', '-d',
                            type=functools.partial(argparse_utils.ranged_float, min_val=0.,
max_val=1.), default=0.,
                            help='drop this percent of the top data (type: float in [0; 1], default:
%(default)s)',
                            metavar='FLOAT')
    data_group.add_argument('--train_valid_test_split_ratio', '-r',
                            type=functools.partial(argparse_utils.ranged_float, min_val=0.,
max_val=1.),
                            default=0.7,
                            help=('split the extracted data into train, validation and test sets with
this ratio'
                                  ' (the number means train percentage, validation and test are
equally splitted from the rest)'
                                  ' (type: float in [0; 1], default: %(default)s)'),
                            metavar='FLOAT')
    data_group.add_argument('--use_spacy_tok_and_lemma', '-spacy',
                            type=str, choices=['en', 'ru', 'none'], default='none',
                            help=('use spaCy tokenizer and lemmatizer for this language model'
                                  ' (type: %(type)s, default: %(default)s)'))

    other_group = parser.add_argument_group('Other', 'Other arguments.')
    other_group.add_argument('--help', '-h', action='help',
                             help='show this help message and exit')
```

```
    return parser


def extract_clean_title(title: str) -> str:
    slash, hyphen, dash = [fr'(?<!\[[^\]]*|\([^\)]*)\s+\{symbol}\s+' for symbol in ('/', '-',
'—')]

    clean_title = regex.split(slash, title)[0] or title
    if clean_title == title:
        clean_title = regex.split(hyphen, title)[0] or title
        if clean_title == title:
            clean_title = regex.split(dash, title)[0] or title

    clean_title = clean_title.lstrip('.:/!,#-' + string.whitespace).rstrip()

    if clean_title[-1] not in ('.', '?', '!'):
        clean_title += '.'

    return clean_title


def extract_clean_code(code: str) -> str:
    brackets = ('(', ')'), ('[', ']'), ('"', '"'), ('«', '»'), ('{', '}'), ('“', '”'), ('<<', '>>')
    brackets_have_been_removed = True

    while brackets_have_been_removed:
        brackets_have_been_removed = False

        for left_bracket, right_bracket in brackets:
            pattern = fr'\{left_bracket}[^{left_bracket}{right_bracket}]*\{right_bracket}'
            code_without_brackets = re.sub(pattern, '', code)

            if code_without_brackets == code:
                brackets_have_been_removed |= False
            else:
                brackets_have_been_removed = True
                code = code_without_brackets

    and_parts = re.split(r'(?:\+|\s+and\s+)', code)

    for and_part_index, and_part_code in enumerate(and_parts):
```

```
        colon_parts = re.split(r':{1,2}', and_part_code)

        for colon_part_index, colon_part_code in enumerate(colon_parts):
            code = re.sub(r'\*.*', '', colon_part_code)
            code = re.sub(r'-.*', '', code)
            code = re.sub(r'=.*', '', code)

            code = re.sub(r'[(\["«{”]|<<.*', '', code)
            code = code.translate(str.maketrans('', '', ')]"»}”'))
            code = code.replace('>>', '')

            code = re.sub(r'[^\d.+/:].*', '', code)

            colon_parts[colon_part_index] = code.strip('.+/:')

        and_parts[and_part_index] = ':'.join(filter(None, colon_parts))

    return '+'.join(filter(None, and_parts))


def generate_taxonomy(labels_trie: datrie.BaseTrie) -> str:
    main_labels = [label for label in labels_trie if len(labels_trie.prefixes(label)) == 1]

    child_taxonomies = [generate_label_taxonomy(label, labels_trie) for label in
main_labels]

    return 'Root\t' + '\t'.join(main_labels) + '\n' + '\n'.join(filter(None, child_taxonomies))


def generate_label_taxonomy(label: str, labels_trie: datrie.BaseTrie) -> str:
    taxonomy = ''

    if len(labels_trie.suffixes(label)) != 1:
        taxonomy = '\t'.join(labels_trie.keys(label))

        last_processed_child = ''

        for child in labels_trie.keys(label)[1:]:
            if last_processed_child not in labels_trie.prefixes(child):
                child_taxonomy = generate_label_taxonomy(child, labels_trie)

                if child_taxonomy:
```

```
                taxonomy += '\n' + child_taxonomy

            last_processed_child = child

    return taxonomy


def extract_tokens(text: str, nlp: typing.Optional[spacy.Language] = None) -> list[str]:
    if not nlp:
        return [token.lower()
                    for token in filter(lambda t: t and t not in string.punctuation,
                        re.split(r'[!"#$%&\'()*+,—\-\\./:;<=>?@[\]^_`{|}~\s]+', text))]
    else:
        return [token.lemma_.lower() for token in nlp(text) if not token.is_punct and not
token.is_stop]


def split_data(split_ratio: float, data: dict[str, list[str]], seed: int) \
        -> tuple[list[str, list[str]], list[str, list[str]], list[str, list[str]]]:
    data = list(data.items())
    random.seed(seed)
    random.shuffle(data)

    train_data_size = int(split_ratio * len(data))
    valid_data_size = (len(data) - train_data_size) // 2

    return data[:train_data_size], data[train_data_size:train_data_size + valid_data_size],
data[
                                                        train_data_size +
valid_data_size:]


def main(cmd_kwargs: dict[str]) -> None:
    data = compress_pickle.load(cmd_kwargs['extracted_data_file'],
set_default_extension=False)

    label_data: dict[str, set[str]] = dict()

    for record in tqdm.tqdm(data):
        clean_title = extract_clean_title(record[cmd_kwargs['data_key']])
        clean_code = extract_clean_code(record[cmd_kwargs['label_key']])
```

```
        if clean_title and clean_code:
            for label in clean_code.split('+'):
                label_data.setdefault(label, set()).add(clean_title)

    labels_trie = datrie.BaseTrie(alphabet=string.digits + '.+/:')
    labels_trie.update(sorted((label, len(data)) for label, data in label_data.items()))

    for label, data in label_data.items():
        for x in data:
            for additional_label in labels_trie.iter_prefixes(label):
                if x not in label_data[additional_label]:
                    label_data[additional_label].add(x)
                    labels_trie[additional_label] += 1

    top_label_data: dict[str, list[str]] = dict()

    for label, label_data_size in heapq.nlargest(cmd_kwargs['top'], labels_trie.items(),
operator.itemgetter(1)):
        top_label_data[label] = list(label_data[label])[:int((1. - cmd_kwargs['drop']) *
label_data_size)]

    top_labels_trie = datrie.BaseTrie(alphabet=string.digits + '.+/:')
    top_labels_trie.update(sorted((label, len(data)) for label, data in
top_label_data.items()))

    data_labels: dict[str, list[str]] = dict()

    for top_label, data in top_label_data.items():
        for x in data:
            for label in top_labels_trie.iter_prefixes(top_label):
                data_labels.setdefault(x, []).append(label)

    for x, y in data_labels.items():
        data_labels[x] = sorted(set(y))

    with open(cmd_kwargs['preprocessed_data_dir'] / 'data.taxonomy', 'w') as file:
        file.write(generate_taxonomy(top_labels_trie) + '\n')

    train_data, valid_data, test_data =
split_data(cmd_kwargs['train_valid_test_split_ratio'], data_labels, 42)

    if cmd_kwargs['use_spacy_tok_and_lemma'] == 'en':
```

```
        nlp = en_core_web_md.load()
    elif cmd_kwargs['use_spacy_tok_and_lemma'] == 'ru':
        nlp = ru_core_news_md.load()
    else:
        nlp = None

    for data, name in (train_data, 'train'), (valid_data, 'valid'), (test_data, 'test'):
        json_data = '\n'.join(json.dumps({'doc_label': y,
                                          'doc_token': extract_tokens(x, nlp),
                                          'doc_keyword': [],
                                          'doc_topic': []}, ensure_ascii=False)
                              for x, y in tqdm.tqdm(data))

        with open(cmd_kwargs['preprocessed_data_dir'] / f'data_{name}.json', 'w') as file:
            file.write(json_data + '\n')


if __name__ == '__main__':
    main(vars(create_cmd_argparser().parse_args()))
```