Санкт-Петербургский государственный университет

Кафедра информационно-аналитических систем

Группа 20.Б08-мм

Рабош Артём Леонидович

Сегментатор листингов для валидатора выпускных квалификационных работ

Отчёт по учебной практике
в форме «Производственное задание»

Научный руководитель:
ассистент Чернышев Г.А.

Санкт-Петербург
2024

Оглавление

Введение

Одним из аспектов университетского образования является написание курсовых и дипломных работ. Данные работы требуют строгого соблюдения формальных требований. Однако проверка соответствия указанным стандартам является трудоемким процессом и требует значительных временных затрат со стороны студентов и преподавателей. Это поднимает вопрос о необходимости автоматизации данного процесса.

Mundane Assignment Police (MAP) — веб-приложение для проверки текстов учебных практик и ВКР. На данный момент MAP поддерживает нахождение различных ошибок оформления в PDF-файлах, таких как, например, неправильный порядок секций.

Одной из особенностей курсовых работ является наличие фрагментов кода — листингов. Важно отметить, что листинги значительно отличаются от обычного текста. Они обычно выделены в отдельные блоки или рамки, имеют специфическое форматирование, а также могут содержать номера строк или другие индикаторы структуры кода. В связи с этим, для листингов устанавливаются отдельные формальные требования, отличные от тех, которые применяются к остальному тексту работы. Такое различие делает важным отделение между текстовыми и кодовыми элементами при автоматической проверке учебных работ.

В настоящее время в приложении MAP отсутствует возможность разделения строк на текстовые и кодовые сегменты. Это приводит к ложным срабатываниям правил проверки и ухудшает пользовательский опыт. В связи с этим, в рамках данной работы предлагается разработать систему, способную проводить сегментацию текста и выделять листинги. Это позволит ограничить применение определенных правил проверки к каждой области, что в свою очередь может значительно снизить число ложных срабатываний и улучшить качество проверки.

1. Постановка задачи

Целью данной работы является разработка системы для сегментации листингов. Для достижения этой цели были поставлены следующие задачи:

1. провести обзор существующих решений для сегментации листингов;
2. разработать архитектуру системы;
3. подготовить набор данных;
4. провести апробацию реализованной системы и сравнить с существующими решениями.

2. Обзор

Исходная задача сегментации листингов представляет собой выделение блоков текста, которые являются листингами. В этом контексте требуется определить для каждой строки, является ли она частью листинга или содержит обычный текст. Важно отметить, что обычный текст может содержать технические элементы, такие как вызовы функций, названия переменных и другие артефакты программирования. В контексте MAP требуется отделять только фрагменты, выделенные в блоки. Эта задачу можно свести к бинарной классификации строк.

В данной главе описываются основные подходы к решению задачи сегментации листингов.

2.1. Методы основанные на эвристиках

В статье «Extracting Source Code from E-Mails» [1] рассматривается задача извлечения фрагментов кода из электронных писем. В работе используется набор эвристик. Например, одной из них является подсчет вхождений специальных символов, таких как точка с запятой, апострофы, кавычки. Также используются регулярные выражения, которые позволяют извлекать синтаксические конструкции, похожие на код.

Важно отметить, что в рассматриваемых письмах код представлен только на языке Java, что сужает область поиска и делает подбор эвристик более узким. Но это позволяет упростить настройку алгоритма и достичь хороших результатов в данной узкой задаче.

Однако, задача извлечения кода из работ студентов, с которой мы сталкиваемся, имеет свои особенности. Код может быть искажен в процессе парсинга PDF, а также он не ограничен языком Java, что усложняет процесс извлечения. Тем не менее, подход, предложенный в указанной статье, может служить важным исходным пунктом для разработки эвристик и методов для нашей задачи извлечения кода из PDF-файлов.

В статье «A Lightweight Approach to Uncover Technical Information in Unstructured Data» [4] участки кода определяются с использованием

различных регулярных выражений. Подход, описанный в статье, рассматривает большее количество языков программирования. Основное внимание уделено выявлению так называемых «технических артефактов» в неструктурированных данных. Артефакты могут включать в себя не только полноценные листинги, но и другие технические элементы, такие как переменные и вызовы функций в тексте, что не соответствует нашей задаче.

2.2. Методы основанные на машинном обучении

В статье «Using Sentence-level Classification Helps Entity Extraction from Material Science Literature» [6] авторы исследуют извлечение информативных предложений из текста научных статей, посвященных материаловедению. Под информативными предложениями в данном контексте понимаются те, которые содержат код, термины, методы или параметры метода. Для извлечения таких предложений из текста используется библиотека Spacy, которая предоставляет функционал для обработки естественного языка.

Важно отметить, что информативные предложения, помеченные как код, не обязательно являются листингами. Это может быть вызвано тем, что подход авторов ориентирован на извлечение предложений, содержащих техническую информацию, независимо от того, представлена ли она в форме листинга или нет.

Для классификации предложений авторы применяют классические методы машинного обучения. Для классических методов в качестве признаков используют векторизацию предложения с помощью tf-idf. Этот метод учитывает частоту слов в предложении и обратную частоту слов во всем корпусе. Кроме того, помимо классических методов, в статье также представлены результаты использования сверточных нейронных сетей (CNN), рекуррентных нейронных сетей (LSTM) и модели BERT.

Полученные авторами метрики говорят о хороших результатах данного подхода. Наилучшие значения метрик получили методы глубоко-

го обучения. Однако это не подходит для нашей задачи по нескольким причинам:

- В предложенном методе классифицируются именно предложения. Листинги кода могут не определятся как предложения в целом. К тому же в MAP текст анализируется построчно, а восстановление строки по извлеченному предложению может быть трудной задачей.

- Как и в предыдущих работах, авторы считали предложения кодом, если в нем есть какая-либо техническая информация. В нашем случае мы хотим классифицировать предложения с такой информацией как текст.

- Модели глубокого обучения требуют вычислительную мощность. В контексте MAP, хочется получить более легкую модель. К тому же авторы не приводят полученных метрик для моделей классического обучения.

3. Набор данных

Для достижения поставленной цели требовалось подготовить набор данных, содержащий работы студентов. Для этого был использован ранее собранный датасет, состоящий из работ в формате PDF, полученных с кафедры системного программирования Санкт-Петербургского государственного университета[1].

Для извлечения текста в инструменте MAP используется парсер PDFBox. Полученные строки подвергаются обработке с помощью определенных правил. Стоит отметить, что PDFBox может вносить изменения в строки текста, поэтому для получения более точных результатов нужно использовать данные, которые получены именно с помощью парсера PDFBox.

Работы были конвертированы в текстовые файлы. Итоговый набор данных содержал в себе 259 тысяч строк из 421 работы. Данные будут использоваться в качестве основы для разработки будущей системы.

3.1. Разметка данных

Во всех текстах были выделены блоки, перед которыми присутствовали маркеры, соответствующие листингам: «Листинг», «Listing», «Алгоритм», «Псевдокод» или «Программа». Однако было замечено, что не все фрагменты кода были явно подписаны. В связи с этим была создана отдельная тестовая выборка, состоящая из 50 последних работ. Строки этих работ были помечены как листинги, если в pdf-файле они были выделены в отдельные блоки и содержали код, вывод программы или псевдокод.

Таблица 1: Результаты разметки

Выборка	Число работ	Число листингов	число строчек кода
Обучающая	371	472	5459
Тестовая	50	65	2372

[1] se.math.spbu.ru (дата обращения: 21.02.2024).

По полученным данным в таблице 1 можно заметить, что на тестовой выборке соотношение числа строк текста к числу строк кода составляет примерно 12:1. Однако, в обучающей выборке это соотношение значительно выше и составляет около 42:1. Это может указывать на то, что часть кода в обучающей выборке не была правильно размечена. Полная ручная разметка обучающей выборки представляется сложной задачей, поэтому в данном случае необходимо работать с имеющимися данными, принимая это ограничение во внимание.

3.2. CodeParrot

Набор текстов работ студентов содержит небольшое число строк, содержащих код. Это может привести к слабой обобщающей способности при использовании моделей машинного обучения. Поэтому было решено найти набор данных, который содержит большое количество строк кода.

Для этого отлично подошёл датасет CodeParrot [5], содержащий открытый код с сайта GitHub. Учитывая значительный размер CodeParrot, требовалось иметь возможность корректно создавать подвыборку. Процесс формирования подвыборки осуществлялся следующим образом:

1. вычисляется распределение файлов по языкам программирования;

2. все файлы принадлежащие одному языку случайно конкатенируются;

3. в каждом языке выбирается нужное число строк с сохранением предварительно вычисленного распределения;

Важно отметить, что был сохранен порядок строк кода, поскольку в будущей модели предполагалось использовать признаки из соседних строк. Все строчки из датасета CodeParrot отмечаются как листинг.

Таким образом мы получили возможность корректно семплировать необходимое количество данных содержащих код. В будущем мы будем

подмешивать полученные данные к работам студентов при обучении моделей. При этом количество строк из CodeParrot было решено оставить в качестве гиперпараметра CodeParrotSize (соотношение числа добавленных строк к числу строк исходной обучающей выборки). В отличии от датасета, содержащего работы студентов, данные из CodeParrot не были предварительно обработаны парсером PDF. Это может повлиять на качество модели, поэтому нужно принимать эту особенность во внимание.

4. Описание реализации

В данной части работы приводится описание разработанной системы, некоторых её особенностей.

При выборе метода реализации было несколько альтернатив:

- использование правил, основанных на регулярных выражениях и эвристиках;
- использование алгоритма классификации на основе машинного обучения.

Было решено реализовать модель на основе машинного обучения, а также базовую модель на основе одного из подходов, предложенных в обзоре, и сравнить результаты.

Для реализации были выбраны язык программирования Python и фреймворк машинного обучения Sklearn.

4.1. Базовая модель

В качестве базовой модели была выбрана система предложенная в статье [4]. Предлагается анализировать текст построчно и определять, содержит ли данная строка технические артефакты или нет. Для этого используется следующий набор критериев:

- CamelCase — наличие подстрок в стиле CamelCase;
- LanguageKeywords — наличие ключевых слов, часто встречающихся в языках программирования;
- SpecialCharacters — наличие специальных символов.

Авторы не приводят списка ключевых слов. Поэтому для создания этого списка было решено использовать датасет CodeParrot. Из датасета был собран частотный словарь терминов и на основе него были выбраны 500 наиболее часто встречающихся слов.

Критерии наличия подстроки в стиле CamelCase и наличия символов пунктуации были реализованы с помощью регулярных выражений. Итоговая система определяет строчку как листинг при выполнении хотя бы одного критерия.

4.2. Модель машинного обучения

4.2.1. Признаки строк на основе эвристик

Для каждой строки были вычислены различные категориальные и числовые признаки, которые можно использовать в модели машинного обучения. Список подсчитанных признаков приведен ниже:

- is_link — наличие ссылки;
- is_comment — наличие комментария, характерного для кода;
- is_attribute — поиск подстрок формата “объект.атрибут”;
- num_of_words — количество слов в строке;
- num_of_words_ru — количество слов на кириллице;
- num_of_words_en — количество слов на латинице;
- ru_percantage — соотношение числа слов на кириллице к общему количеству слов в строке;
- en_percantage — соотношение числа слов на латинице к общему количеству слов в строке;
- brackets — наличие подстроки вида “()”;
- num_index_letter — количество подстрок вида [a], где a — любая латинская буква;
- num_index_word — количество подстрок вида [x], где x — число;
- len — длина строки.

4.2.2. Векторизация строк

Для извлечения текстовых признаков был выполнен процесс токенизации строк. Проведено сравнение различных токенизаторов, и наилучшее качество достигнуто с использованием собственного токенизатора. Принцип его работы основан на следующих шагах: вначале из полученной строки извлекаются все знаки пунктуации, после чего выделяются все буквенные последовательности. Каждая из этих последовательностей затем обрабатывается с использованием стеммера (в данном случае PORTERStemmer для русского и английского языков из библиотеки NLTK), что позволяет получить набор токенов. Все числовые значения в строке заменяются на токен “NUMBER”.

В процессе векторизации текста применяется метод “Мешок слов” (Bag-of-Words). Этот метод основан на подсчете количества вхождений слов в каждую строку текста и последующем построении матрицы признаков. В отличие от метода tf-idf, который учитывает в основном уникальные слова, Bag-of-Words рассматривает слова не обращая внимания на частоту по всему корпусу текста. Подсчёт матрицы текстовых признаков осуществляется с помощью класса CountVectorizer из библиотеки Scikit-learn.

Так же для необработанной каждой строчки были посчитаны дополнительные признаки перечисленные в 4.1 и 4.2.1.

Ввиду последовательной структуры текста, целесообразным является использование информации из предыдущих и последующих строк в качестве дополнительных признаков. Для этого было введено два гиперпараметра: NumOfPreviousLines и NumOfNextLines, определяющие количество строк, взятых из предыдущего и последующего контекста соответственно. Полученные векторы строк контекста конкатенируются с вектором классифицируемой строки. Важно учитывать, что слишком большое значение этих параметров может привести к переобучению модели. Кроме того, учитывая значительно меньшее число дополнительных признаков были создан гиперпараметр — NumOfPreviousStatistics, отвечающий за использование данных признаков с последних n

строчек.

4.2.3. Выбор модели машинного обучения

Для классификации с помощью машинного обучения были выбраны следующие модели:

- Логистическая регрессия
- Градиентный бустинг
- FastText [2]

Отдельно стоит отметить FastText. FastText это фреймворк для классификации текстов от компании FaceBook. Он обучается напрямую на корпусе текстов, без необходимости предварительной векторизации. В связи с этим, мы не можем использовать дополнительные признаки вместе с этой моделью.

С точки зрения модели машинного обучения FastText представляет из себя нейронную сеть с одним скрытым полносвязным слоем. Для векторизации текста он может использовать модели skipgram и cbow (continuous-bag-of-words). Так же FastText предоставляет возможность автоматического подбора гиперпараметров для заданной метрики на валидационной выборке.

Для обучения модели логистической регрессии использовалась библиотека Scikit-learn. Градиентный бустинг обучался с помощью известного open -source фреймворка CatBoost [3].

4.2.4. Подбор параметров и критерии оценивания

В процессе оценки качества классификаторы обычно применяются метрики, основанные на матрице ошибок, представленной на рисунке 1. TP обозначает количество верно классифицированных положительных примеров (true positives), FP — количество ложно положительных примеров (false positives), TN — количество верно классифицированных отрицательных примеров (true negatives), а FN — коли-

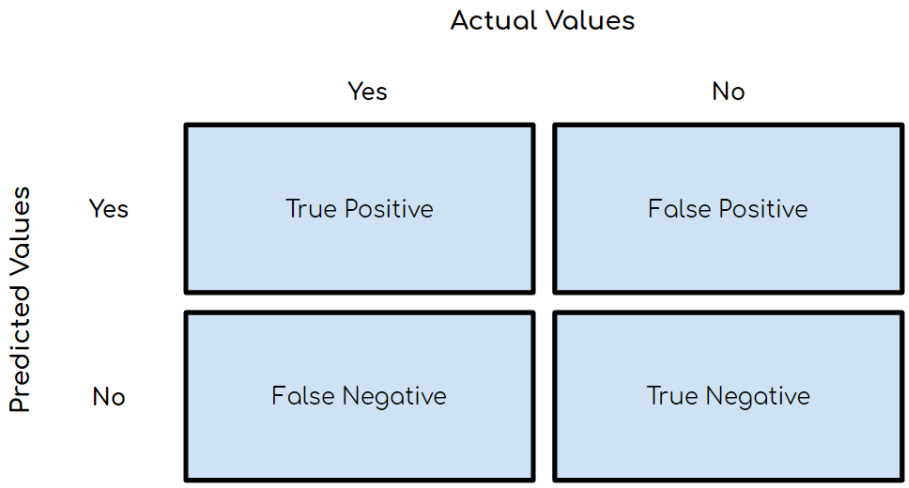

Рис. 1: Матрица ошибок

чество ложно отрицательных примеров (false negatives). Для оценки качества классификации мы будем использовать следующие метрики:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

В контексте инструмента MAP была поставлена задача снижения числа ложных срабатываний на строках, содержащих листинг. Это означает, что полученная модель должна определять наибольшее число строк кода. В терминах метрик задачу можно поставить, как максимизация метрики *Recall*. Поэтому при подборе гиперпараметров целевой метрикой мы будем считать именно её. Поскольку мы можем варьировать метрики за счёт изменения порога бинаризации (threshold), мы должны зафиксировать значение метрики *Precision*, для того, чтобы число ошибочно классифицированного текста было не слишком много. Было решено зафиксировать *Precision* на уровне 0.5. Во фреймворке FastText это можно сделать с помощью параметра autotunePredictions.

Для всех моделей осуществлялся подбор параметров по сетке. Каж-

дому параметру задан набор возможных значений. Перебирая все комбинации параметров и замеряя метрики, находим наилучшую в своём классе модель.

Ниже представлен набор возможных значений параметров по которым проводился подбор. Список гиперпараметров данных и процесса векторизации с возможными значениями для перебора представлен ниже (для модели FastText используется только CodeParrotSize). Гиперпараметры для модели логистической регрессии и градиентного бустинга представлены в таблице 2 и 3 соответственно. Курсивом выделены параметры показавшие наилучшие целевые метрики на тестовой выборке.

- CodeParrotSize — 0.0, 0.5, **1.0**, 1.5, 2.0
- NumOfPreviousLines — 0, 1, **2**, 3, 4
- NumOfNextLines — 0, **1**, 2, 3, 4
- NumOfPreviousStatistics – 0, 1, 2, **5**, 10

Таблица 2: Гиперпараметры логистической регрессии

Параметр	Значение
fit intercept	**True**, False
solver	**lbfgs**, liblinear, saga
C	**0.1**, 0.2,0.5, 1.0
penalty	**l1**, l2

Таблица 3: Гиперпараметры градиентного бустинга

Параметр	Значение
iterations	**1000**
depth	3, 4, **5**, 6
l_2 leaf reg	0.0, **0.1**, 0.2, 0.5

5. Эксперимент

Для начала были получены метрики при использовании базового метода:

Таблица 4: Метрики правил на тестовой выборке

Правило	Accuracy	Precision	Recall
CamelCase	0.84	0.17	0.13
ProgrammingKeywords	0.63	0.17	0.81
Special characters	0.64	0.13	0.48
Combined	0.53	0.14	0.84

Базовая модель позволяет определяет 84% строк содержащих код, однако при этом слишком велико «потерянных» строк текста. Так же важно заметить, что при использовании базовой модели невозможно варьировать параметры для достижения необходимого значения метрики *Precision*. Ниже представлены метрики с использованием моделей машинного обучения.

Таблица 5: Метрики на тестовой выборке

Правило	Precision	Recall
Логистическая регрессия	0.5	0.88
Градиентный Бустинг	0.5	0.94
FastText	0.5	0.78

Все модели на основе машинного обучения показали значительно лучшие результаты по сравнению с базовой моделью. Наилучшие значения целевой метрики показал градиентный бустинг на текстовых и дополнительных признаках. FastText не учитывает символы пунктуации в тексте, к тому же к нему нельзя добавить свои признаки. Это могло стать причиной относительно плохого качества классификации с помощью этого метода.

6. Заключение

В рамках работы были достигнуты следующие результаты:

- Проведен обзор решений для задачи сегментации листинга.
- Реализована система для сегментации листинга.
- Подготовлен набор данных для тестирования и обучения систем.
- Проведено сравнение реализованной системы с существующими решениями.

В дальнейшем планируется использовать полученные результаты в разработке расширенного сегментатора текста, позволяющего сегментировать не только листинги, но и таблицы. Открытый код проекта доступен по ссылке[2] на репозиторий Github.

[2] https://github.com/artyomrabosh/practice_map — репозиторий на Github (дата обращения: 21.02.2024).

Список литературы

[1] Bacchelli Alberto, D'Ambros Marco, Lanza Michele. Extracting Source Code from E-Mails // 2010 IEEE 18th International Conference on Program Comprehension. — 2010. — P. 24–33.

[2] Bag of Tricks for Efficient Text Classification / Armand Joulin, Edouard Grave, Piotr Bojanowski, Tomas Mikolov // arXiv preprint arXiv:1607.01759. — 2016.

[3] CatBoost: unbiased boosting with categorical features / Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev et al. // Proceedings of the 32nd International Conference on Neural Information Processing Systems. — NIPS'18. — Red Hook, NY, USA : Curran Associates Inc., 2018. — P. 6639–6649.

[4] A Lightweight Approach to Uncover Technical Artifacts in Unstructured Data / Nicolas Bettenburg, Bram Adams, Ahmed E. Hassan, Michel Smidt // 2011 IEEE 19th International Conference on Program Comprehension. — 2011. — P. 185–188.

[5] Tunstall L., von Werra L., Wolf T. Natural Language Processing with Transformers: Building Language Applications with Hugging Face. — O'Reilly Media, 2022. — ISBN: 9781098103248. — URL: https://books.google.ru/books?id=pNBpzwEACAAJ.

[6] Using Sentence-level Classification Helps Entity Extraction from Material Science Literature / Ankan Mullick, Shubhraneel Pal, Tapas Nayak et al. // Proceedings of the Thirteenth Language Resources and Evaluation Conference / Ed. by Nicoletta Calzolari, Frédéric Béchet, Philippe Blache et al. — Marseille, France : European Language Resources Association, 2022. — June. — P. 4540–4545. — URL: https://aclanthology.org/2022.lrec-1.483.