Санкт-Петербургский государственный университет

Математическое обеспечение и администрирование информационных систем

Нафикова Лиана Ирековна

Обработка таблиц в приложении для валидации текстов выпускных квалификационных работ

Учебная практика

Научный руководитель:
ассистент кафедры ИАС Чернышев Г. А.

Санкт-Петербург
2023

Оглавление

Введение

В процессе написания ВКР соблюдение определенных требований к форматированию и оформлению текста играет важную роль. Текст работы должен быть тщательно проверен несколько раз — как самим студентом, так и его научным руководителем. Проверка соответствия стандартам оформления требует таких ресурсов, как внимание и время. При этом нет необходимости в ручной проверке отступов, шрифтов, расположения изображений, если эта задача поддаётся автоматизации.

Для облегчения проверки соответствия формальным требованиям было создано приложение Mundane Assignment Police, написанное на языке Kotlin. Веб-приложение обрабатывает работы, загруженные в формате PDF, и подчёркивает строки, в которых студент допустил ошибки. На данный момент система способна находить неправильное использование тире и дефисов, скобок, обнаруживать некорректное написание чисел, ссылок, цитат и др.

Очень часто студенты для более наглядного представления исходных данных и результатов тестирования используют в своих работах таблицы. Таблицы требуют отдельной обработки, так как имеют отличную от обычного текста структуру. Структура текста описана в [8]. Данные же в таблице обычно представляются в виде полей и записей. При этом на каждое поле может быть наложено определённое требование при оформлении. Например, для чисел с плавающей точкой необходимо следить за тем, чтобы количество десятичных знаков в колонке было одинаково. Выявление таблиц предоставит возможность определять зоны текста, а это, в свою очередь, уменьшит количество ложных срабатываний, так как можно будет отключать и добавлять новые правила для отдельной зоны.

Поэтому в данной работе предлагается расширить функциональность приложения Mundane Assignment Police, интегрировав в него модуль для работы с таблицами.

1. Постановка задачи

Целью данной работы является интегрирование в приложение модуля, обрабатывающего таблицы.

Для её выполнения были поставлены следующие задачи:

1. провести обзор существующих технологий для экстракции таблиц и выбрать лучшую;
2. выполнить доработку выбранного алгоритма извлечения таблиц;
3. интегрировать полученное решение в проект;
4. создать отдельный структурный элемент «Таблица», с помощью которого будет осуществляться обработка всех таблиц;
5. произвести тестирование и апробацию решения.

2. Обзор

На данный момент в проекте для обработки PDF-документов используется библиотека PDFBox, которая не даёт существенных результатов по обработке таблиц: если с извлечением текста библиотека справляется довольно неплохо, то табличная структура не сохраняется вовсе. Поэтому было решено выбрать другую существующую открытую библиотеку и в случае необходимости доработать её. Основными критериями выбора являются автоматическое обнаружение табличных данных в тексте и точность обнаружения. Такие библиотеки как iText [5], Aspose.PDF [2], Spire.PDF [3] имеют ряд ограничений на количество обрабатываемых листов, либо не могут автоматически находить таблицы в тексте. Поэтому было решено сравнить такие библиотеки как Camelot [1], tabula-py [7].

2.1. Camelot

Camelot, библиотека, написанная на Python, даёт возможность гибко изменять настройки извлечения таблиц. Каждая таблица извлекается в pandas DataFrame, возможен экспорт в несколько форматов, таких как CSV, JSON, HTML и другие. Внутри Camelot возможен выбор двух методов извлечения таблиц: Stream и Latice.

Для реализации алгоритма Latice в Camelot используется библиотека The Poppler PDF [4] для рендеринга изображений PDF-документа. После рендеринга изображение преобразуется в черно-белый формат. Алгоритм обнаружения границ таблицы работает на анализе интенсивности пикселей и её разностью между соседними пикселями. Библиотека даёт возможность выбирать пороговое значение интенсивности пикселей границы. Но стоит иметь в виду, что при выборе слишком высокого порогового значения некоторые из более тонких визуальных подсказок на странице не будут обнаружены, а слишком низкое пороговое значение может привести к большому количеству ошибочно интерпретируемых границ таблиц. Далее отдельно находятся горизонтальные рёбра, вертикальные, точки пересечения и отдельные ячейки.

Алгоритм обнаружения горизонтальных границ таблицы начинается с

изучения пикселей изображения в оттенках серого из верхнего левого угла. Сравнивается каждая пара соседних пикселей сверху вниз, ищутся изменения значения интенсивности выше установленного порогового значения. Когда найдена пара пикселей с достаточной разницей в их интенсивности, алгоритм переходит вправо, сравнивая несколько пикселей (в направлении вверх-вниз), пока край не исчезнет или не встретится правый край изображения. Если найденное ребро имеет достаточную длину, оно принимается и регистрируется как горизонтальное ребро на изображении. Процесс обнаружения вертикальных границ аналогичен. После нахождения границ отдельные пиксели, например, диапазона в 10 пикселей друг от друга, усредняются, и это среднее значение становится «точкой пересечения границ» с центром в одном пикселе. Последним этапом процесса обнаружения границ является идентификация замкнутых прямоугольных пространств внутри вертикальных и горизонтальных разделительных линий и разделение несоединенных прямоугольных областей на разные таблицы.

Stream можно использовать для анализа таблиц с пробелами между ячейками для имитации структуры таблицы. Так как анализа видимых границ таблицы этот алгоритм не предполагает, главной его задачей является отделение табличных элементов от нетабличных. Для реализации предлагается:

1. удалить все текстовые элементы с полей (например, номера страниц);
2. разделить текстовые элементы на строки;
3. обнаружить границы текстовых прямоугольных блоков;
4. отделить табличные прямоугольные блоки от выровненного по ширине текста;
5. ранжировать строки по признаку вероятности принадлежности к таблице;
6. назначить прямоугольные области;
7. обработать прямоугольные области (например, обнаружить разделение по столбцам, строкам, найти заголовки и т.д.)

Шестой шаг процесса обнаружения таблиц включает в себя определение пределов таблиц. Если на странице существуют сетки и определенные прямоугольные области, найденные алгоритмом, описанным выше, то они классифицируются как таблицы. При отсутствии сеток строки, определенные как содержащие табличное содержимое, объединяются в прямоугольные области.

Более подробно алгоритмы представлены в работе [6].

2.2. Tabula

Tabula — ещё один инструмент, позволяющий читать таблицы из PDF. В качестве алгоритмов извлечения Tabula также использует алгоритмы, описанных в [6]. У библиотеки, написанной на Java, есть оболочки Ruby[1], Python[2], R[3], and Node.js[4], позволяющие ускорить процесс извлечения большого количества таблиц из PDF-файла. Разработка на Ruby, Node.js более не ведётся.

2.3. Выбор библиотеки

Также разработчики провели сравнение[5] Camelot с такими библиотеками, как Tabula, pdfplumber, pdftables, pdf-table-extract. Camelot работает лучше, чем Tabula, во всех случаях, где таблицы явно отрисованы. Tabula лучше обнаруживает таблицы для случаев Stream. Pdfplumber не идентифицирует несколько таблиц на странице, PDFTables и pdf-table-extract не справляются с таблицами без видимых границ и объединёнными колонками. Для PDFTables приостановлена open-source разработка.

[1]https://github.com/tabulapdf/tabula-extractor — репозиторий библиотеки tabula-extractor на Github (дата обращения: 15.12.2023).

[2]https://github.com/chezou/tabula-py — репозиторий библиотеки tabula-py на Github (дата обращения: 15.12.2023).

[3]https://github.com/ropensci/tabulizer — репозиторий библиотеки tabulizer на Github (дата обращения: 15.12.2023).

[4]https://github.com/ezodude/tabula-js — репозиторий библиотеки tabula-js на Github (дата обращения: 15.12.2023).

[5]https://github.com/camelot-dev/camelot/wiki/Comparison-with-other-PDF-Table-Extraction-libraries-and-tools — сравнение Camelot с другими библиотеками на Github (дата обращения: 15.12.2023).

Из-за особой структуры PDF-документа студенческой работы нецелесообразно использовать алгоритм, основанный на пробелах. Например, качестве таблицы обнаруживается титульный лист, так как он имеет особого рода табуляции, соответствующие формату. Поэтому в качестве используемого алгоритма было решено использовать Latice, а в качестве реализующей библиотеки Camelot.

3. Реализация

Latice не поддерживает анализ текста, поэтому, чтобы алгоритм обнаружил таблицу, необходимо наличие замкнутой прямоугольной области на странице. Поэтому таблицы, в которых не хватает части рёбер не будут обнаружены. Чтобы повысить количество найденных таблиц, предлагается дорисовать недостающие рёбра. Различаются три основных случая расположения рёбер:

1. на странице присутствуют только горизонтальные рёбра;
2. на странице присутствуют только вертикальные рёбра;
3. на странице есть отдельные связанные табличные области, в каждой из которых находятся либо таблицы из только горизонтальных или только вертикальных рёбер, либо составленные из горизонтальных и вертикальных рёбер таблицы, но при этом незамкнутые.

Каждая линия представляет кортёж четырёх чисел: (x1, y1, x2, y2), где (x1, y1) — координаты начала линии, (x2, y2) — координаты конца.

3.1. Горизонтальные рёбра на странице

Пример такого расположения представлен на данной странице:

Таблица 1: Горизонтальные рёбра

text	text
53.492	77.906
46.753	61.587

Так как анализ текста не предполагается, считается, что линии, выравненные по левому краю (т.е. по x1), принадлежат одной таблице. Сначала

происходит сортировка рёбер по x1, затем по y1. Среди рёбер с общей левой координатой x1 ищется соответственно минимальные и максимальные y1 и x2. Затем с помощью библиотеки cv2 (данная библиотека используется Camelot для рендеринга изображения) отрисовывается прямоугольная область с максимальной площадью, содержащая в себе все необходимые рёбра.

В результате для Таблицы 2 получается следующая корректировка:

Таблица 2: Скорректированные рёбра

text[%]	text [%]
53.492	77.906
46.753	61.587

3.2. Вертикальные рёбра

В случае, когда на странице только вертикальные рёбра, корректировка выполняется похожим образом. Рёбра сортируются сначала по верхнему краю (т.е. по y2), потом по левому (x1). Затем линии, находящиеся на одной высоте помещаются в прямоугольную область с максимальной по всем рёбрам длиной и шириной, равной ширине листа. Пример корректировки представлен ниже:

Таблица 3: Вертикальные рёбра

heading 1	**heading 2**	**heading 3**
α	β	γ
1	11.34	a
2	10.5	b
3	765.5231	c

Таблица 4: Скорректированные рёбра

heading 1	heading 2	heading 3
α	β	γ
1	11.34	a
2	10.5	b
3	765.5231	c

3.3. Связанные табличные области

В более сложном случае на странице присутствует несколько таблиц, каждая из которых может состоять как и из вертикальных рёбер, так и из горизонтальных. Поэтому появилась необходимость отделить друг от друга табличные области и для каждой выполнить необходимую дорисовку.

Будем считать, что пересекающиеся рёбра принадлежат одной табличной области. Для каждой пары рёбер проверяется пересечение, в случае положительного ответа сегменты, их содержащие, объединяются. В результате получается список сегментов и тех рёбер, которые не пересекаются с другими.

Для каждого из сегментов аналогично находятся минимальные и максимальные x1, x2, y1, y2. Вся таблица помещается в заданную прямоугольную область.

Таблица 5: Связанная табличная область

Value 1 α	**Value 2** β	**Value 3** γ
1234		a b
3	23.113 231	c
4	25.113 231	d

Несвязанные рёбра также делятся на горизонтальные и вертикальные, и для каждой группы выполняется дорисовка, аналогичная описанным в пунктах 3.1 и 3.2.

Таблица 6: Скорректированные рёбра

Value 1 α	**Value 2** β	**Value 3** γ
1234		a b
3 4	23.113 231 25.113 231	c d

3.4. Интеграция функциональности в проект

Основное приложение и модуль написаны на разных языках. Поэтому необходимо было обеспечить их совместимость. Есть несколько способов сделать это:

- использование Jython;
- отдельный сервер Python, взаимодействующий с сервером Kotlin;
- использование скрипта Python вместо отдельного проекта.

Jython — реализация Python, написанная на Java. Основной минус его использования — многие библиотеки его не поддерживают. Сервер же не удобен из-за того, что вызовет задержку. Поэтому было решено использовать скрипт для экстрации таблиц в csv-файлы, а потом считать данные из файлов внутри Kotlin.

Для Python необходимо установить виртуальное окружение со всеми необходимыми библиотеками. Чтобы запустить скрипт, создаётся отедельный процесс. Внутри скрипта во время считывания таблиц, также с помощью Camelot вытаскивается служебная информация, которая будет использоваться в дальнейшем. В эту служебную информацию входят номер страницы, координаты таблицы, координаты отдельных ячеек, количество полей и записей. Они также записываются в csv-файл, хранящий данные таблицы.

3.5. Создание структурного элемента для обработки таблиц

Фрейм данных — это удобная абстракция для работы со структурированными данными. Абстрацию делает удобной набор операций, определенных в ней. А Kotlin DataFrame[6] — это язык на основе Kotlin для определения таких операций. Он используется для считывания табличных данных из csv-файлов. Считанный DataFrame является полем в структурном элементе «Таблицы». Также полями являются страница, границы таблицы, количество колонок и строк.

Также все «Таблицы» имеют список «Клеток». Для каждой клетки аналогично с помощью службной информации фиксируется страница, координаты левого нижнего угла, правого верхнего угла, и строки текста, находящиеся внутри. Весь текст внутри таблицы обрабатывается схожим с обычным текстом образом: отдельные символы являются структурными единицами слова, а слова — структурными единицами строки. Отличие лишь в том, что теперь строкой являются не все слова, находящиеся на одной высоте, а те слова, которые находятся внутри определённой ячейки на фиксированной высоте. Нахождение слова внутри ячейки опять-таки возможно проверить с помощью служебной информации таблицы.

[6]https://github.com/Kotlin/dataframe — репозиторий библиотеки Kotlin DataFrame на Github (дата обращения: 15.12.2023).

4. Эксперименты

Тестирование проводилось на базе 30 работ[7] Математико-механического факультета СПбГУ. Для каждой работы проверяется количество найденных таблиц и время их обработки. Результаты представлены в Таблице 7.

[7]https://github.com/Darderion/map-dataset — репозиторий с базой выпускных квалификационных работ на Github (дата обращения: 15.04.2023).

Таблица 7: Табличные области в работах студентов

Тип работы	Количество таблиц	Количество обнаруженных таблиц	Время обработки, с	Количество страниц
Курсовые	1	9	9.369	13
	6	6	20.259	22
	6	0	12.085	17
	8	8	16.930	21
	5	4	20.229	27
	10	16	9.446	14
	9	8	16.283	17
	1	1	10.353	14
	5	6	20.521	28
	2	2	24.436	32
Бакалаврские	14	18	41.468	41
	10	18	25.541	38
	14	13	28.581	33
	4	4	17.690	19
	15	14	22.771	49
	16	15	33.734	31
	8	8	14.928	26
	15	5	22.906	39
	14	11	26.028	32
	10	15	29.075	49
Магистерские	30	31	22.559	30
	20	14	39.896	56
	3	3	25.264	33
	13	12	34.153	47
	52	52	48.855	63
	9	15	26.466	37
	10	8	26.069	30
	21	26	29.155	41
	9	5	18.264	26
	30	42	35.649	48

Эксперименты показали довольно медленную скорость выполнения. Это связано в первую очередь с генерацией изображения каждой отдельной страницы. Также нетрудно заметить, что количество прямоугольных областей не всегда совпадает с полученным в результате экстракции.

Первая причина заключается в том, что алгоритм библиотеки, используемый в данной работе, распознаёт именно прямоугольные области (т.е. не обязательно таблицы), находящиеся на странице. Важно заметить, что для любой такой области (например, схемы) не требуется проверять правила, подходящие для валидации обычного текста. То есть графики и диаграммы тоже рассмотрены, как табличные элементы.

Помимо этого Camelot предоставляет возможность варьировать параметры экстракции. Именно они определяют, будет ли рассмотрена прямоугольная область как таблица. В частности, напрямую на экстракцию влияет параметр *line_scale*. Он является коэффициентом масштабирования размера линии. Если линия слишком узка относительно размера текста, то она не будет распознана.

Также бывают таблицы, которые состоят из нескольких несвязанных областей:

Таблица 8: Одна таблица из двух несвязанных областей

Тип ошибки	Алгоритм	0-315
rmse	mrob	8.347
	orb3	**7.822**
mean	mrob	**5.474**
	orb3	6.846
median	mrob	**5.094**
	orb3	6.091

Заключение

Были выполнены следующие задачи:

1. проведён обзор существующих технологий для экстракции таблиц и выбрана лучшая;
2. выполнена доработка выбранного алгоритма
3. произведено интегрирование полученного решения в проект;
4. реализовано создание отдельного структурного элемента «Таблица», с помощью которого будет осуществляться обработка всех таблиц в работах;
5. произведены тестирование и апробация решения.

Открытый код проекта доступен по ссылке на репозиторий[8] Github.

[8]https://github.com/Liana2707/map — репозиторий проекта на Github (дата обращения: 15.10.2022).

Список литературы

[1] Camelot. — https://github.com/camelot-dev/camelot. — Accessed: 15.12.2023.

[2] Documentation for Aspose.PDF. — https://docs.aspose.com/pdf/java/. — Accessed: 15.12.2023.

[3] Documentation for Spire.PDF. — https://www.e-iceblue.com/Introduce/pdf-for-java.html. — Accessed: 15.12.2023.

[4] Documentation for The Poppler PDF. — https://poppler.freedesktop.org. — Accessed: 15.12.2023.

[5] Lowagie Bruno. iText in Action: Covers iText 5. — Manning, 2010.

[6] Nurminen Anssi. Algorithmic extraction of data in tables in PDF documents : Master's thesis ; Tampere University of Technology. — Tampere University, Kalevantie 4, 33100, Tampere, 2013.

[7] tabula-py. — https://github.com/chezou/tabula-py. — Accessed: 15.12.2023.

[8] Хмельницкая Е.В. Методические указания по оформлению выпускной квалификационной работы. — 2018.