

РЕФЕРАТ

Отчет 79 с., 49 рис., 12 табл., 33 источн.

КЛАССИФИКАЦИЯ, DOCX, ODT, PDF, СТРУКТУРНЫЕ ЭЛЕМЕНТЫ, CATBOOST, RANDOMOVERSAMPLING, CONFUSION MATRIX, SKLEARN, PDFPLUMBER, TABULA, УНИФИЦИРОВАННЫЕ КЛАССЫ

Объектом исследования является система автоматизированного нормоконтроля документов

Цель работы – автоматизация процесса нормоконтроля отчетной документации в формате PDF для уменьшения количества времени проверки и исключения человеческого фактора.

В процессе работы: проводились поиск и анализ существующих решений, исследованы различия между ODT, DOCX, PDF форматами текстовых документов в разных категориях, выделены и описаны проблемы существующих решений и проблемы возникающие при парсинге PDF документов, описаны разработанные алгоритмы для парсинга структурных элементов и их свойств из PDF документов, рассмотрены разработанные унифицированные классы структурных элементов, описаны созданный классификатор на основе CatBoost и реализованный модуль GateWay Api для создания единой точки входа в сервис и взаимодействия всех модулей проекта.

В результате исследования были:

- разработана библиотека парсинга PDF документов, Реализован набор унифицированных классов, представляющих структурные элементы,
- разработан Web-сервис автоматизированного нормоконтроля документации,
- разработаны алгоритмы парсинга структурных элементов pdf документов и их свойств,
- реализована модель для анализа структурных элементов.

СОДЕРЖАНИЕ

| | |
|---|----|
| ВВЕДЕНИЕ | 4 |
| 1 Существующие решения и их анализ | 6 |
| 1.1 Обзор аналогов | 6 |
| 1.2. Исследование различий в структуре и способах хранения информации о контенте в различных типах форматов текстовых документов | 11 |
| 1.2.1 Различия в структуре PDF документов от ODT и DOX форматов файлов | 11 |
| 1.2.2 Различия в хранении контента структурных элементов | 19 |
| 1.2.3 Различия в хранении свойств и параметров структурных элементов | 21 |
| 1.2.4 Различия в хранении табличных объектов | 25 |
| 1.2.5 Различия в хранении изображений | 27 |
| 1.3 Проблемы парсинга PDF документов | 30 |
| 1.4 Вывод по главе 1 | 33 |
| 2. Алгоритмы для извлечения структурных элементов | 34 |
| 2.1 Алгоритмы для строк | 35 |
| 2.2 Алгоритмы для извлечения текстовых параграфов | 37 |
| 2.3 Алгоритмы для извлечения свойств текстовых параграфов | 39 |
| 2.4 Алгоритмы для извлечения рисунков и таблиц | 41 |
| 2.5 Вывод по главе 2 | 44 |
| 3. Модификация веб-сервиса | 46 |
| 3.1 Унифицированные классы структурных элементов | 46 |
| 3.2 Классификатор текстовых структурных элементов документа | 51 |
| 3.2.1 Описание существующего решения | 51 |

| | |
|---|----|
| 3.2.2 Проектирование и оценка модели классификации текстовых структурных элементов документов..... | 56 |
| 3.3 Api Gateway..... | 60 |
| 3.3.1 RabbitMQ | 61 |
| 3.3.2 GateWay..... | 63 |
| 3.3.3 RPC Server..... | 66 |
| 3.3.4 Результаты работы сервиса..... | 69 |
| 3.4 Вывод по главе 3 | 72 |
| ЗАКЛЮЧЕНИЕ | 73 |
| СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ | 75 |

ВВЕДЕНИЕ

Решение задачи извлечения свойств и параметров текстов выпускных квалификационных работ, научных диссертаций необходимо для проведения автоматизированного нормоконтроля документов. Множество научных и студенческих работ имеют ошибки в оформлении и могут не соответствовать ГОСТам и другим стандартам различных Университетов. Выделение данных о параграфах и фрагментах текста в дальнейшем может быть использовано для классификации элементов и поиска ошибок и несоответствий требованиям стандартов. Автоматизация данного процесса может существенно сократить время, затрачиваемое преподавателями, либо другими лицами, отвечающими за нормоконтроль, и соответственно снизить расходы на проверку.

Согласно статистике, большинство научных работ, таких как отчеты по научно-исследовательским работам, выпускные квалификационные работы, и т. д. хранятся и распространяются в виде текстовых документов формата PDF.

Portable Document Format (PDF) - формат, разработанный компанией Adobe в качестве окончательного, результирующего формата текстовых документов, который не будет меняться в зависимости от версий текстового редактора [1]. Из-за этого данный формат идеально подошёл для передачи, распространения и печати документов. Изначально предполагалось только минимальная возможность редактирования содержимого. Однако на данный момент существуют более продвинутое расширения формата, например тегирование, но они не обрели такого массового распространения как оригинал [2].

Для обеспечения различных видов обработки структурных элементов текстовых документов необходимо разработать метод извлечения параграфов, изображений и таблиц и их свойств.

В последние несколько десятилетий исследователями были проведены работы по извлечению и сегментации текста, различных способах обработки изображений, выделению таблиц. Большая часть данных исследований касаются газетных статей и журналов.

Множество программных решений для парсинга PDF позволяют извлекать только текст и изображения. Некоторые из них - отдельные свойства текста. Тем не менее, ни один из вариантов библиотек не подходит для определения корректности оформления структурных элементов документа. Однако существующие решения могут быть использованы для получения свойств и объектов PDF документа.

Цель работы - автоматизация процесса нормоконтроля отчетной документации в формате PDF для уменьшения количества времени проверки и исключения человеческого фактора.

Задачи работы:

- исследовать различия между форматами текстовых документов по разным категориям,
- изучить аналоги и существующие решения,
- исследовать проблемы, возникающие при парсинге PDF документов,
- разработать алгоритмы для парсинга различных структурных элементов текстовых документов формата PDF,
- адаптировать сервис “Автоматизированного нормоконтроля документов” под работу с различными форматами текстовых документов, включая PDF.

1 Существующие решения и их анализ

1.1 Обзор аналогов

В нынешнюю эпоху происходит постоянный процесс глобализации. Это подразумевает собой сведение каждой вещи к определенному стандарту или норме. Соответственно и оформление научных отчетов, статей, книг и тому подобное сводится к единому виду. Поэтому процесс проверки оформления текстовых документов или если по-другому нормоконтроль приобретает всё более и более важное значение. Этот процесс происходит преимущественно вручную и в некоторых случаях с большим процентом ошибок.

Для определения текущего состояния проблемы был проведён анализ существующих решений. Он показал, что ранее исследователи уже проводили попытки автоматизировать процесс нормоконтроля. В ходе исследования были выделены наиболее успешные попытки решения этой проблемы.

В статье «Обеспечение качества конструкторской документации за счет средств автоматизации нормоконтроля» [3] рассматривается способ увеличения качества конструкторской документации при помощи автоматизации процесса нормоконтроля, т. е. проверки рабочих документов на соответствие ГОСТам и другим нормативным актам. Представляемая система проверяет документацию в электронном виде и только определённых форматов документов. Поддержка только определённого формата документов не всегда является удобным пользователям. В статье приводится результат выявления и анализа существующих ГОСТов и нормативов, касающихся конструкторской документации. Также авторы описывают спроектированную базу данных, содержащую стандартную документацию и решения для дальнейшего использования пользователями. В своей работы авторы утверждают, что внедрение данной системы экономит трудозатраты, повышает качество документации за счёт существенного сокращения ошибок.

Авторы статьи [4] описывают разработанный метод автоматизированного нормоконтроля техдокументации. В результате исследования авторы утверждают, что благодаря спроектированному методу

уменьшаются временные затраты на процесс нормоконтроля и уменьшается количество пропущенных ошибок. В статье утверждается, что нынешняя система нормоконтроля неэффективна и может быть автоматизирована. Авторы ссылаются на то, что ГОСТы и другие нормативные акты содержат и описывают правила оформления текста и других элементов таким образом, что их нетрудно запрограммировать при помощи шаблонов или параметров проверки. Таким образом за счёт шаблонизации правил и других инструментов процесс нормоконтроля проходит существенно легче и быстрее.

В статье [5] поднимается вопрос важности нормоконтроля документов при помощи автоматизированных средств. Целью работы авторов было создание алгоритма для проверки дипломных работ, хранящихся в виде текстовых документов, на соответствие ГОСТам. Авторами был предложен алгоритм, в основе которого лежат циклы. Первый цикл проводит проверку всех параграфов текста на правильность оформления отступов, интервалов и других свойств, и параметров. Второй цикл, находящийся внутри первого, проводит похожую проверку стилей оформления всех слов в параграфе. И соответственно третий цикл, лежащий в теле второго, проверяет все символы в слове. В результате его работы обнаруживаются ошибки, которые не соответствуют представляемым ГОСТам. Представленный алгоритм реализован в виде приложения, разработанного при помощи языка программирования C#. Программа поддерживает работу с документами форматов DOC, DOCX, DOT и некоторых других. В приложении существует несколько режимов его работы: режим для обучения алгоритма путём добавления эталонных образцов и последующего сравнения проверяемых документов с ними, режим «Редакторский» в результате работы которого выводится протокол проверки, хранящий в себе данные об ошибках и отклонениях.

В статье «Некоторые особенности поэтапного алгоритма программы для проверки дипломных работ на нормоконтроль» [6] описывается алгоритм работы программы, отвечающей за проведение нормоконтроля дипломных

работ студентов. Однако данный алгоритм работает только с определённым форматом документов, поддерживаемый MS Word - DOCX и соответственно написан на языке программирования VBA. Программа предоставляет функционал для: нормоконтроля документов DOCX, указания полученных сведений с указанием даты, предоставление подробного отчёта об ошибках и возможностях их исправлений.

Существует мобильное приложение, «Электронный нормоконтролёр» [7], выполняющее, как заверяют авторы, функцию проверки оформления дипломных работ на соответствия ГОСТу 7.32–2001. Приложением поддерживается только формат документов DOCX. Оно имеет достаточно простой интерфейс – достаточно загрузить документ и нажать кнопку проверить. Однако в связи с тем, что оно поддерживает работу только на мобильном устройстве имеет место неудобность его использования, так как текстовые документы в основном создают и заполняют на персональном компьютере. При попытке проверки точности работы системы возникла проблема с тем, что оно не поддерживает современные версии операционных систем мобильных устройств. Кроме этого, из оценок и комментариев пользователей было выяснено что у множества из них приложение не запустилось.

В рамках НИРМ в 2019–2021 годах разрабатывался проект «Сервис автоматизированного нормоконтроля документов и обучения оформлению документации», основной функцией которой является проверка научных диссертаций на правильность оформления в соответствии с определённым ГОСТом [8].

Приложение основано на использовании клиент-серверной архитектуры. В связи с тем, что чаще всего при написании каких-либо отчётов и работ используется программа MS Word, то клиентская часть базируется на технологии Microsoft Office Add-In technology, что позволяет напрямую в программе, не используя внешние средства, сделать проверку оформления.

Серверная часть использует технологию RestFul API для получения запросов от клиентской части и соответственно отправки ответов обратно.

Спроектированная система работает следующим образом:

- авторизация пользователя,
- прикрепление документа и его загрузка на сервер,
- извлечение необходимых данных и их преобразование, сохранение в CSV формате при помощи модуля DocxCorrector,
- поэлементная классификация текста,
- согласование результата с пользователем,
- в случае несогласия с результатами пользователь может исправить ошибочные результаты,
- проверка правильности оформления элементов согласно ГОСТу.

Программа имеет три режима работы: «нормоконтролёр», «ленивый студент» и «интерактивно».

Режим «Нормоконтролёр» проводит проверку оформления документа и определяет его соответствие ГОСТу, неправильные участки текста он выделяет и выводит пользователю. Данный режим позволяет пользователю сверять соответствие определённых системой классов параграфов.

Режим «Интерактивно» является режимом по умолчанию и проходит весь цикл работы системы.

Режим «Ленивый студент» выполняет полную проверку документа, находит несоответствующие участки и исправляет их. В данном случае пользователь не может исправить классы параграфов и таким образом полностью доверяет системе. Пользователь получает уже исправленный документ.

Модуль DocxCorrector разработан на базе технологии .NET Core и выполняет две основные функции:

- выделение (парсинг) свойств и параметров параграфов текста для их последующей классификации,

– проверка оформления документа на соответствие ГОСТам.

Классификатор использует алгоритм градиентного бустинга, основанный на открытой технологии CatBoost, разработанной компанией «Яндекс». Класс элемента определяется на основе его свойств и параметров.

Главной проблемой всех представленных приложений является работа с ограниченным количеством форматов документов, в основном только с DOCX документами.

Следует учитывать, что большинство из поддерживаемых форматов являются закрытыми и для их чтения требуются специальные лицензионные программы. В последние года политика РФ направлена на импортозамещение, что значит использование бесплатных, открытых или отечественных разработок, что может предполагать в будущем замещение форматов MS WORD на формат ODT или ODF.

Очень часто происходит так что при открытии одного и того же документа на различных устройствах оформление документа может существенно измениться, что в некоторых ситуациях может быть весомой проблемой. На практике студенты и преподаватели для обмена документами используют документы форматов PDF в связи с тем, что независимо от устройства контент внутри документа не изменить своё оформление.

Например, известное веб-приложение Антиплагиат позволяет использовать исключительно форматы PDF и TXT для проведения проверки на антиплагиат.

Эти проблемы может существенно уменьшить количество пользователей. Для её решения предлагается создать функционал для работы с форматом PDF. Для этого необходимо сравнить его с используемым форматами, определить, где можно адаптировать существующие алгоритмы, где необходимо создать новые и что самое главное – какие данные можно извлечь и проанализировать.

1.2. Исследование различий в структуре и способах хранения информации о контенте в различных типах форматов текстовых документов

Для понимания как сильно и в чём различаются форматы их необходимо сравнить по разным категориям. В ходе работы сравнение происходило в следующих категориях:

- хранение текста,
- хранение свойств и параметров,
- хранение таблиц и их содержимого,
- хранение изображений.

В связи с использованием одинаковых технологий форматы ODT и DOCX рассматривались вместе там, где различия не существенные. [1, 9]

1.2.1 Различия в структуре PDF документов от ODT и DOCX форматов файлов

PDF (Portable Document Format) – кроссплатформенный формат электронных документов, разработанный компанией Adobe Systems с использованием ряда возможностей языка PostScript. В первую очередь предназначен для представления печатной продукции в электронном виде. Значительное количество современного профессионального печатного оборудования имеет аппаратную поддержку формата PDF, что позволяет производить печать документов в данном формате без использования какого-либо специального программного обеспечения [1].

Формат PDF позволяет внедрять необходимые шрифты, растровые и векторные изображения, формы и мультимедийные вставки. Включает механизм электронных подписей для защиты и проверки целостности и подлинности документов.

PDF документ представляет собой бинарный файл, который можно разделить на следующие 4 части, как показано на рисунке 1 [1,9]:

- заголовок,

- тело,
- xref таблица,
- прицеп.

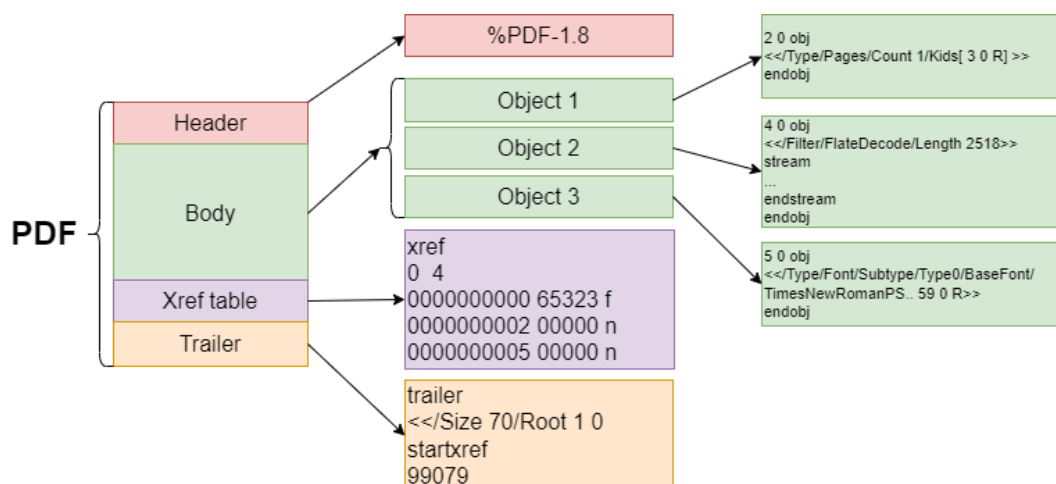


Рисунок 1 – Структура PDF документа

В заголовке документа описывается только номер версии спецификации используемого pdf формата. В зависимости от версии считывающая программа будет по-разному расшифровывать файл.

Тело документа содержит все объекты и типы данных, находящиеся в файле и поддерживаемые текущей спецификацией PDF. Они могут включать в себя текст, картинки, таблицы и многое другое.

Xref Таблица является одной из основных отличительных особенностей данного формата и представляет собой строки содержащие данные о расстоянии определённого объекта от начала файла в байтах, таким образом программе открывающей pdf файл не обязательно загружать весь документ сразу, а можно открыть только представленную пользователю страницу. На рисунке 4 представлена Xref таблица документа PDF.

Прицеп содержит данные о местоположении xref таблицы в файле и основном, родительском объекте документа. Иногда документ может содержать несколько прицепов.

В конце PDF документа всегда присутствует строка %%EOF.

Одной из отличительных особенностей pdf является то, что чтение файла начинается от конца файла, с прицепа.

Типы данных, поддерживаемые PDF документом.

PDF документ поддерживает множество типов данных, которые используются в теле объекта – объекты, массивы, строки, словари, потоки.

Строки представляют собой последовательность восьмибитных символов, заключённых в круглые скобки. При создании строк следует учитывать, что символ обратного слэша не является частью строки, а переносит последующий после него текст на следующую строку, либо экранирует следующий после него символ.

Текстовая информация в строке может храниться несколькими способами: привычным символьным способом, с помощью восьмеричного кода символа, например \010, с помощью специальных двух-байтовых HEX-символов, например <7253105F>.

Массивы представляют собой последовательность других типов данных, чаще всего текстовых строк, которая заключена в квадратные скобки. Чаще всего массивы содержат в себе текст. В массиве PDF документа, содержащего в себе текст, между его элементами очень часто можно увидеть различные положительные и отрицательные числа. Данные числа отвечают за индивидуальное позиционирование предыдущих символов на странице документа. Чаще всего они появляются при преобразовании других форматов, например DOCX в PDF документ. Пример массива представлен на рисунке 2.

A screenshot of a PDF array, which is a sequence of objects enclosed in square brackets. The array contains: an object 'o', a negative integer '-5', an empty space ' ', a positive integer '9', an object 'Wo', a negative integer '-6', and a string 'rld' enclosed in single quotes.

Рисунок 2 – Тип данных – массив

Словари представляют собой данные вида ключ – значение. В основном в документах формата PDF они используются для предания объектам и другим типам данных определённых свойств, например длину потока в байтах. Все словари в документе должны быть заключены в треугольные скобки в начале словаря и в конце соответственно. Пример словаря представлен на рисунке 3.

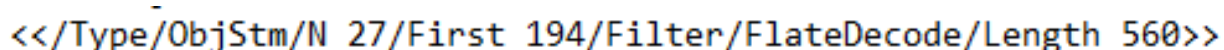
A screenshot of a PDF dictionary, which is a set of key-value pairs enclosed in double angle brackets. The dictionary contains: a key-value pair with key '/' and value 'Type', a key-value pair with key '/N' and value '27', a key-value pair with key '/First' and value '194', a key-value pair with key '/Filter' and value 'FlateDecode', and a key-value pair with key '/Length' and value '560'.

Рисунок 3 – Тип данных – словарь

Поток представляет собой последовательность восьмибитных данных, заключённых между ключевыми словами stream и endstream. Поток содержит в себе сжатые данные текста, картинки, шрифта и многое другое. Поток обязательно должен иметь свойства, описанные при помощи словаря, такие как длина, метод сжатия и т. п.

Объект — является основным типом данных PDF документа. Он может включать в себя множество других типов, описывающих картинки текст и другие данные. Каждому объекту PDF файла присваивается собственный ID, который однозначно идентифицирует конкретный объект, как показано на рисунке 4. Внутренние данные объекта могут также ссылаться на другой объект, описывающий, например шрифт.

```
22 0 obj
<</Type/ObjStm/N 27/First 194/Filter/FlateDecode/Length 560>>
stream
хъкUЫj11}}дхt_нВruBK€19Ў;||I-ьКЦ«°^СьчЕльн-нэITМим™>гъ .@1a||eA+`J /fS х||Aq3A||E||,fд||У μF;(Y,P *r||P(<]f;||D
U%f+тU|= сибђu|| f Ж%~
` S2K*Кц||ЧJ||ghb||ooИДf(ЛЙбLИьшjЙ-п||«юЎ±;тг ь||Ид||Вu||№i||Sц||ы ||OА°Ы||€НВt,,Мi' i||0f||НqЙ+ё°ыC\<3Г°6||C\8`||8M||дSД1ы||ХТК:c[
endstream
endobj
```

Рисунок 4 – Объект в PDF документе

Приведённый пример содержит в себе как словарь, так и сжатый поток. Из словаря можно понять, что поток состоит из 560 байт, что он сжат (/Filter) с помощью gzip (/FlateDecode). Эти данные позволяют разжать поток и получить хранящиеся в нём данные.

Поток PDF файла может быть сжат различными способами, но в основном используются gzip, ASCII Hex, ASCII 85-based, LZW. Также дополнительно он может быть зашифрован.

1.2.1.2 DOCX документ

DOCX (Microsoft Word Open XML Document) — документы, созданные программным обеспечением для обработки текста Microsoft Word. Данный формат разработан компанией Microsoft. Файл с расширением DOCX содержит текст, изображения, форматирование текста, нарисованные объекты и другие элементы. Чаще всего применяется для создания деловых, научных и

личных документов, является одним из самых популярных форматом среди текстовых редакторов.

DOCX файлы спроектированы так, чтобы содержимое текстового документа было доступным — текстовый документ сохраняется с помощью текстовых файлов (plain text) и изображений документов хранятся в виде отдельных файлов в DOCX формате.

В то время как PDF документ хранит данные в одном единственном бинарном файле, DOCX документ, создаётся с помощью Open XML и представляет собой архив, содержащий папки и файлы. Архив DOCX документа состоит из 3 папок: `_rels`, `docProps`, `word`, содержащих XML файлы. XML файлы хранят в себе контент, свойства и параметры контента, метаданные документа и взаимосвязи между файлами. На рисунке 5 представлена структура DOCX документа.

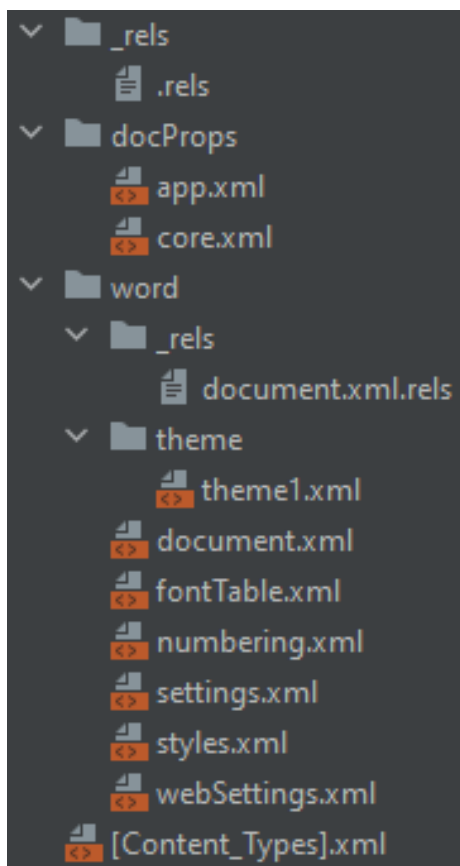


Рисунок 5 – Структура DOCX документа

Файл `_rels/.rels` содержит информацию о структуре документа. Он хранит информацию о местоположении файлов с метаданными и основному

документу, содержащему контекст. На рисунке 6 представлен пример файла rels/.rels.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
- <Relationships xmlns="http://schemas.openxmlformats.org/package/2006/relationships">
  <Relationship Target="docProps/app.xml" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/extended-properties"
    Id="rId3"/>
  <Relationship Target="docProps/core.xml" Type="http://schemas.openxmlformats.org/package/2006/relationships/metadata/core-properties"
    Id="rId2"/>
  <Relationship Target="word/document.xml" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/officeDocument"
    Id="rId1"/>
</Relationships>
```

Рисунок 6 – Пример файла rels/.rels

Документ DOCX хранит свои метаданные как правило в папке docProps. Обычно два файла core.xml, представленный на рисунке 7, и app.xml хранят информацию об авторе и времени создания и последнего изменения, наименовании приложения, создавшего документ, версии документа и т. п. соответственно.

```
<?xml version="1.0" encoding="UTF-8" standalone="true"?>
<Properties xmlns:vt="http://schemas.openxmlformats.org/officeDocument/2006/docPropsVTypes"
  xmlns="http://schemas.openxmlformats.org/officeDocument/2006/extended-properties">
  <Template>Normal.dotm</Template>
  <TotalTime>1240</TotalTime>
  <Pages>1</Pages>
  <Words>87</Words>
  <Characters>499</Characters>
  <Application>Microsoft Office Word</Application>
  <DocSecurity>0</DocSecurity>
  <Lines>4</Lines>
  <Paragraphs>1</Paragraphs>
  <ScaleCrop>>false</ScaleCrop>
  - <HeadingPairs>
    - <vt:vector baseType="variant" size="2">
      - <vt:variant>
        <vt:lpstr>Название</vt:lpstr>
      </vt:variant>
      - <vt:variant>
        <vt:i4>1</vt:i4>
      </vt:variant>
    </vt:vector>
  </HeadingPairs>
  - <TitlesOfParts>
    - <vt:vector baseType="lpstr" size="1">
      <vt:lpstr/>
    </vt:vector>
  </TitlesOfParts>
  <Company/>
  <LinksUpToDate>>false</LinksUpToDate>
  <CharactersWithSpaces>585</CharactersWithSpaces>
  <SharedDoc>>false</SharedDoc>
  <HyperlinksChanged>>false</HyperlinksChanged>
  <AppVersion>16.0000</AppVersion>
</Properties>
```

Рисунок 7 – Пример файла core.xml

Папка Word хранит фактическое содержание документа. Файл XML, названный document.xml, является основным документом, содержащий большую часть текста самого документа. На рисунке 8 представлен фрагмент файла document.xml.


```

<w:body>
- <w:p w:rsidP="00C51F50" w:rsidRDefault="00C51F50" w:rsidR="001B2A75" w14:textId="68F422B5" w14:paraId="4DAE6ACF">
- <w:pPr>
  <w:pStyle w:val="a3"/>
  <w:numPr>
    <w:ilvl w:val="0"/>
    <w:numId w:val="1"/>
  </w:numPr>
</w:pPr>
- <w:r>
  <w:t>Исследование возможности использования нейросетей для систем распознавания лиц.</w:t>
</w:r>
</w:p>
- <w:p w:rsidP="00C51F50" w:rsidRDefault="00C51F50" w:rsidR="00C51F50" w14:textId="13324527" w14:paraId="78E46B7E">
- <w:pPr>
  <w:pStyle w:val="a3"/>
  <w:numPr>
    <w:ilvl w:val="0"/>
    <w:numId w:val="1"/>
  </w:numPr>
</w:pPr>
- <w:r>

```

Рисунок 8 – Фрагмент файла document.xml

В то же время файл styles.xml содержит информацию о стилях оформления документа. На рисунке 9 представлен список стилей документа DOCX.

```

- <w:style w:styleId="a" w:default="1" w:type="paragraph">
  <w:name w:val="Normal"/>
  <w:qFormat/>
</w:style>
- <w:style w:styleId="a0" w:default="1" w:type="character">
  <w:name w:val="Default Paragraph Font"/>
  <w:uiPriority w:val="1"/>
  <w:semiHidden/>
  <w:unhideWhenUsed/>
</w:style>
- <w:style w:styleId="a1" w:default="1" w:type="table">
  <w:name w:val="Normal Table"/>
  <w:uiPriority w:val="99"/>
  <w:semiHidden/>
  <w:unhideWhenUsed/>
  <w:tblPr>
    <w:tblInd w:w="0" w:type="dxa"/>
    <w:tblCellMar>
      <w:top w:w="0" w:type="dxa"/>
      <w:left w:w="108" w:type="dxa"/>
      <w:bottom w:w="0" w:type="dxa"/>
      <w:right w:w="108" w:type="dxa"/>
    </w:tblCellMar>
  </w:tblPr>
</w:style>
+ <w:style w:styleId="a2" w:default="1" w:type="numbering">
- <w:style w:styleId="a3" w:type="paragraph">
  <w:name w:val="List Paragraph"/>
  <w:basedOn w:val="a"/>
  <w:uiPriority w:val="34"/>
  <w:qFormat/>
  <w:rsid w:val="00C51F50"/>
  <w:pPr>
    <w:ind w:left="720"/>
    <w:contextualSpacing/>
  </w:pPr>
</w:style>

```

Рисунок 9 – Фрагмент файла style.xml

1.2.1.3 ODT документ

OpenDocument Format, ODF (Open Document Format for Office Application — открытый формат документов для офисных приложений) —

открытый формат документов для обмена и хранения офисными документами, текстовыми документами (отчеты, заметки, книги), таблицами, презентациями, рисунками.

Стандарт разработан индустриальным сообществом OASIS и основан на формате XML. 1 мая 2006 года принят как международный стандарт ISO/IEC 26300. Стандарт разработан различными организациями, может использоваться без ограничений и доступен для всех. Представляет собой альтернативу закрытым форматам (DOC, XLS, PPT), а также формату Microsoft Office Open XML (DOCX, XLSX, PPTX).

Аналогично DOCX документу ODT документ представляет собой архив, с вложенными в него папками и XML файлами. В XML файлах содержится как основной текст, так и метаданные, такие как информация об авторе, дате, программе в которой он был создан. Также в отдельных файлах содержится информация о параметрах текста, таких как размер шрифта, его тип, информация о локализации на странице, различные параметры вывода на экран и печати и многое другое.

Основной текст документа и его свойства содержатся в файле, именованном content.xml, часть свойств и параметров текста и списков содержатся в XML файлах style.xml. Также существует специальная папка «media» в которой хранятся внесённые в документ изображения и видео файлы. Структура документа ODT представлена на рисунке 10.

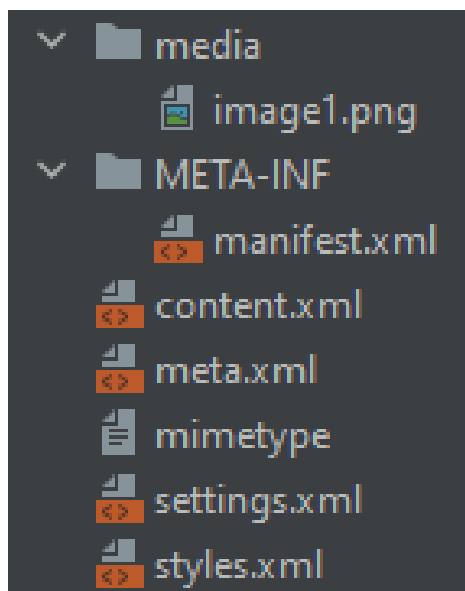


Рисунок 10 – Структура ODT документа

Придание тексту определённого стиля полностью основана на использовании XML тегов, которые имеют понятное как для специалистов, так и для простых пользователей представление. Пример внутренней структуры файла content.xml представлен на рисунке 11.

```
+ <office:font-face-decls>
+ <office:automatic-styles>
- <office:body>
  - <office:text text:use-soft-page-breaks="true">
    - <text:list text:continue-numbering="true" text:style-name="LFO1">
      + <text:list-item>
      + <text:list-item>
      + <text:list-item>
      + <text:list-item>
      - <text:list-item>
        - <text:p text:style-name="P5">
          Исследование возможности использования генеративно-состязательных
          нейронных сетей для
          дипфейков
          <text:s/>
        </text:p>
      </text:list-item>
      + <text:list-item>
    </text:list>
    <text:p text:style-name="P7">11.11.2021</text:p>
    + <text:p text:style-name="Обычный">
    + <text:p text:style-name="Обычный">
      <text:p text:style-name="Обычный"/>
    + <table:table table:style-name="Table10">
    + <text:p text:style-name="Обычный">
  </office:text>
</office:body>
```

Рисунок 11 – Пример внутренней структуры файла content.xml

1.2.2 Различия в хранении контента структурных элементов

Способы хранения текста в DOCX и ODT схожи и достаточно просты. Весь текст в файлах document и content соответственно хранится в виде

последовательности параграфов заключённых в специальные теги: `<w:p>` и `<text:p>` соответственно. Для придания определённого стиля оформления тексту данным тегам присваиваются определённые теги или свойства. Все абзацы объединяются под общим тегом `<w:body>` и `<office:body>` соответственно.

В свою очередь внутри параграфа текст делится на фрагменты, в основном построчно и заключается в теги `<w:r>` в DOXС документе. В ODT документе строки разделяются посредством добавления между ними тега `<text:s/>`.

Если текст содержит символы или слова из разных языков, то в не зависимости от его расположения в тексте фрагмент с иноязычными символами заключается в тег `<w:r>` в DOCX, либо добавляется теги `<text:s/>`, а сам фрагмент заключается в тег `<text:span>`

Основной текст, специально не сжатый и не зашифрованный пользователем, представляет собой обычную последовательность русских, английских символов алфавита, а также других языков мира и оформляется в тег `<w:t>`. Фрагмент файла document, хранящий одно предложение представлен на рисунке 12.

```
- <w:p w:rsidRDefault="001E779D" w:rsidR="00C83B0F" w14:textId="4BC65C64" w14:paraId="610A766A" w:rsidRPr="001E779D">
  - <w:r>
    <w:t xml:space="preserve">Привет, </w:t>
  </w:r>
  - <w:r>
    - <w:rPr>
      <w:lang w:val="en-US"/>
    </w:rPr>
    <w:t xml:space="preserve">my name is </w:t>
  </w:r>
  - <w:r>
    <w:t>Слава</w:t>
  </w:r>
</w:p>
```

Рисунок 12 – Пример хранения текста в DOCX документе

Таким образом документы DOCX и ODT в способе хранения текста различаются только в наименовании тегов, а в основных принципах практически идентичны.

PDF документ хранит текст чаще всего в сжатом виде в массиве, состоящем из последовательности строк и находящийся внутри потока, который в свою очередь хранится в определённом объекте.

В потоке начало и конец текста отмечаются специальными операторами: BT – begin text и ET – end text. Однако в этом фрагменте кроме самого текста хранится и информация о его местоположении на странице, шрифт, размер и т. п. Сам основной текст отмечен символами TJ после соответствующего массива как показано на рисунке 13.

```
BT
/F1 11.04 Tf
1 0 0 1 85.104 774.84 Tm
/GS7 gs
0 g
/GS8 gs
0 G
[(H)3(e11)] TJ
ET
```

Рисунок 13 – Пример хранения текста в PDF документе

Внутри массива текст также может быть зашифрован в зависимости от первоначальных настроек редактирования текста в текстовом редакторе.

Следует отметить, что в отличие от документов ODT и DOCX контент представлен не в виде параграфов, а в виде групп символов, сгруппированных по их местоположению на странице, т.е. если например одно предложение занимает 2 строки, но первая строка имеет 1,25 см отступ от начала страницы, то эти строки будут находиться в 2 разных фрагментах в потоке или вовсе в разных потоках.

1.2.3 Различия в хранении свойств и параметров структурных элементов

Документ DOCX хранит все свойства и параметры текста в стилях. Список и параметры стиля содержатся в файле style.xml. На рисунке 14 представлен фрагмент файла style.xml, содержащий список с несколькими стилями. Тег <w:style> объявляет стиль в списке стилей. Каждый стиль должен иметь тип и идентификационное имя.

```

- <w:style w:styleId="a3" w:type="paragraph">
  <w:name w:val="List Paragraph"/>
  <w:basedOn w:val="a"/>
  <w:uiPriority w:val="34"/>
  <w:qFormat/>
  <w:rsid w:val="00C51F50"/>
- <w:pPr>
  <w:ind w:left="720"/>
  <w:contextualSpacing/>
</w:pPr>
</w:style>
- <w:style w:styleId="a4" w:type="table">
  <w:name w:val="Table Grid"/>
  <w:basedOn w:val="a1"/>
  <w:uiPriority w:val="39"/>
  <w:rsid w:val="006D5950"/>
- <w:pPr>
  <w:spacing w:lineRule="auto" w:line="240" w:after="0"/>
</w:pPr>
- <w:tblPr>
  - <w:tblBorders>
    <w:top w:val="single" w:color="auto" w:space="0" w:sz="4"/>
    <w:left w:val="single" w:color="auto" w:space="0" w:sz="4"/>
    <w:bottom w:val="single" w:color="auto" w:space="0" w:sz="4"/>
    <w:right w:val="single" w:color="auto" w:space="0" w:sz="4"/>
    <w:insideH w:val="single" w:color="auto" w:space="0" w:sz="4"/>
    <w:insideV w:val="single" w:color="auto" w:space="0" w:sz="4"/>
  </w:tblBorders>
</w:tblPr>
</w:style>

```

Рисунок 14 – Пример хранения свойств и параметров текста в DOCX документе

В свою очередь в файле с основным контекстом в теге параграфа или фрагмента происходит ссылка на существующий стиль. Следует отметить, что в определённых случаях параметры могут быть указаны напрямую в теге объявления текста.

В отличие от DOCX документов, в котором стили расположены в отдельном файле, все стили объектов в формате ODT хранятся вместе с контекстом в файле content.xml. В свою очередь это позволяет не ссылаться на другой файл и обращается напрямую к стилю.

На рисунке 15 представлен фрагмент файла content.xml объявляющий стили текста, списков и таблицы.

```

+ <text:list-style style=name="LF01">
+ <style:style style=name="P1" style=family="paragraph" style:list-style-name="LF01" style:master-page-name="MP0" style:parent-style-name="Абзацсписка">
+ <style:style style=name="P2" style=family="paragraph" style:list-style-name="LF01" style:parent-style-name="Абзацсписка"/>
+ <style:style style=name="P3" style=family="paragraph" style:list-style-name="LF01" style:parent-style-name="Абзацсписка">
+ <style:style style=name="P4" style=family="paragraph" style:list-style-name="LF01" style:parent-style-name="Абзацсписка"/>
+ <style:style style=name="P5" style=family="paragraph" style:list-style-name="LF01" style:parent-style-name="Абзацсписка">
+ <style:style style=name="P6" style=family="paragraph" style:list-style-name="LF01" style:parent-style-name="Абзацсписка">
+ <style:style style=name="P7" style=family="paragraph" style:parent-style-name="Обычный">
+ <style:style style=name="T8" style=family="text" style:parent-style-name="Основнойшрифтабзаца">
+ <style:style style=name="T9" style=family="text" style:parent-style-name="Основнойшрифтабзаца">
+ <style:style style=name="TableColumn11" style=family="table-column">
+ <style:style style=name="TableColumn12" style=family="table-column">
+ <style:style style=name="Table10" style=family="table">
+ <style:style style=name="TableRow13" style=family="table-row">
+ <style:style style=name="TableCell14" style=family="table-cell">
+ <style:style style=name="P15" style=family="paragraph" style:parent-style-name="Обычный">
+ <style:style style=name="TableCell16" style=family="table-cell">
+ <style:table-cell-properties fo:padding-right="0.075in" fo:padding-bottom="0in" fo:padding-left="0.075in" fo:padding-top="0in" style:writing-mode="lr-tb" fo:border="0.0069in solid #000000"/>
+ </style:table-cell-properties>
+ </style:table-cell-properties>
+ <style:style style=name="P17" style=family="paragraph" style:parent-style-name="Обычный">
+ <style:style style=name="TableRow18" style=family="table-row">
+ <style:style style=name="TableCell19" style=family="table-cell">
+ <style:style style=name="P20" style=family="paragraph" style:parent-style-name="Обычный">
+ <style:style style=name="TableCell21" style=family="table-cell">
+ <style:style style=name="P22" style=family="paragraph" style:parent-style-name="Обычный">
+ <style:style style=name="a0" style=family="graphic" style:parent-style-name="Graphics">

```

Рисунок 15 – Пример хранения свойств и параметров текста в ODT документе

Далее в тегах, объявляющих текст документа, прописывается стиль при помощи ссылки на него по его имени.

Хранение свойств в PDF документах имеет значительные отличия по сравнению с другими форматами. Свойства текста задаются посредством использования специальных символов или сочетания символов. Актуальная спецификация PDF поддерживает большое количество свойств и соответственно символов. Основные символы, а также их значение представлены в таблице 1.

Таблица 1 – Основные операторы формата PDF

| Оператор | Значение |
|----------------|--|
| BT...ET | Начало и конец текста |
| Tf | Шрифт |
| Td, TD, Tm, T* | Позиционирование текста |
| Tj | Показать текстовую строку |
| TJ | Показать текстовую строку с учётом индивидуального позиционирования символов |
| TI | Расстояние между строками |
| Tr | Опция рендеринга |
| Tc | Межсимвольное расстояние |
| Tw | Расстояние между словами |

| | |
|----|---|
| Th | Горизонтальное масштабирование символов |
| Ts | Смещение текста по вертикали по сравнению с базовым местоположением |
| Tk | Инверсия текста |
| g | Прозрачность серого символа |
| w | Ширина линии |

Следует отметить, что оператор идёт после задания его значений.

Пример хранения свойств текста представлен на рисунке 16.

```
BT
/F1 11.04 Tf
1 0 0 1 85.104 737.74 Tm
0 g
0 G
[(Erg)4(erg.)] TJ
ET
```

Рисунок 16 – Пример хранения свойств и параметров текста в PDF документе

Такие параметры как жирный, курсивом и тому подобное в основном задаются в отдельном объекте представляющим собой шрифт. Конкретные свойства задаются при помощи типа данных – словарь. На рисунке 17 представлены два объекта представляющие одинаковые шрифты, но с разным написанием: обычным и жирным соответственно.

```
5 0 obj
<</Type/Font/Subtype/TrueType/Name/F1/BaseFont/BCDEEE+Calibri/Encoding/WinAnsiEncoding/FontDescriptor 6 0 R/FirstChar 32/LastChar 119/Widths 24 0 R>>
endobj

9 0 obj
<</Type/Font/Subtype/TrueType/Name/F2/BaseFont/BCDFEE+Calibri-Bold/Encoding/WinAnsiEncoding/FontDescriptor 10 0 R/FirstChar 69/LastChar 114/Widths 26 0 R>>
endobj
```

Рисунок 17 – Пример определения шрифта в PDF документах

1.2.4 Различия в хранении табличных объектов

Для создания и хранения таблиц документы ODT и DOCX используют специальные теги. Свойства таблицы, колонок и ячеек также описываются с помощью тегов, которые могут ссылаться на определённый стиль, либо прописываться напрямую в XML файле. Основные используемые для создания таблицы и придания ей определённого стиля теги описаны в таблице 2.

Таблица 2 – Основные теги, используемы для хранения таблиц

| Тег DOCX | Тег ODT | Значение |
|--------------|----------------------|---------------------------|
| w:tbl | table:table | Тег, объявляющий таблицу |
| w:tblPr | - | Свойства таблицы |
| w:tblStyle | table:style-name | Ссылка на стиль таблицы |
| w:tblW | - | Ширина таблицы |
| w:tblGrid | table:table-columnns | Тег, объединяющий колонки |
| w:tblGridCol | table:style-name | Свойства колонок |
| w:tr | table:table-row | Строка таблицы |
| w:tc | table:table-cell | Ячейка таблицы в строке |

Текст внутри ячеек подчиняется таким же правилам, как и обычный текст вне таблицы, т. е. используются те же самые теги и свойства.

Пример хранения таблицы в DOCX документе представлен на рисунке 18.

```

- <w:tbl>
  - <w:tblPr>
    <w:tblStyle w:val="a4"/>
    <w:tblW w:w="0" w:type="auto"/>
    <w:tblLook w:val="04A0" w:noVBand="1" w:noHBand="0" w:lastColumn="0" w:firstColumn="1" w:lastRow="0" w:firstRow="1"/>
  </w:tblPr>
  - <w:tblGrid>
    <w:gridCol w:w="4672"/>
    <w:gridCol w:w="4673"/>
  </w:tblGrid>
  - <w:tr w:rsidR="006D5950" w14:textId="77777777" w14:paraId="1335A2CE" w:rsidTr="006D5950">
    - <w:tc>
      + <w:tcPr>
        - <w:p w:rsidRDefault="006D5950" w:rsidR="006D5950" w14:textId="6AD3F9C6" w14:paraId="3BF2AF83">
          - <w:r>
            <w:t>Имя</w:t>
          </w:r>
        </w:p>
      </w:tcPr>
    </w:tc>
    + <w:tc>
      </w:tr>
  - <w:tr w:rsidR="006D5950" w14:textId="77777777" w14:paraId="39079893" w:rsidTr="006D5950">
    - <w:tc>
      + <w:tcPr>
        - <w:p w:rsidRDefault="006D5950" w:rsidR="006D5950" w14:textId="7F93D3E1" w14:paraId="0BB36798">
          - <w:r>
            <w:t>Вячеслав</w:t>
          </w:r>
        </w:p>
      </w:tcPr>
    </w:tc>
    + <w:tc>
      </w:tr>
  </w:tbl>

```

Рисунок 18 – Пример хранения таблиц в DOCX документе

В связи с тем, что у документа PDF не существует специального объекта – таблицы, то построение таблицы идёт совершенно другим путём.

Первоначально в потоке прописывается текст ячейки по обычным правилам, т. е. указывается шрифт, размер, местоположение и многое другое. Сразу за фрагментом, объявляющим текст ячейки при помощи заданных координат и специального символа, отвечающего за раскраску области по соответствующим координатам, рисуется ячейка. Ячейка образуется посредством закрашивания областей – граней ячейки, поэтому таких областей достаточно много для одной ячейки.

Для каждой ячейки таблицы такие действия повторяются. На рисунке 19 представлен фрагмент файла, рисующий ячейку таблицы.

```

EMC /Artifact BMC 0 g
79.464 725.26 0.48 0.48 re
f*
79.464 725.26 0.48 0.48 re
f*
79.944 725.26 478.15 0.48 re
f*
558.1 725.26 0.47998 0.48 re
f*
558.1 725.26 0.47998 0.48 re
f*
79.464 711.82 0.48 13.44 re
f*
558.1 711.82 0.47998 13.44 re
f*
EMC q
79.944 697.9 478.15 13.44 re
W* n

```

Рисунок 19 – Пример рисования ячеек таблицы в PDF документе

1.2.5 Различия в хранении изображений

Хранение рисунков и изображений в документах ODT и DOCX происходит достаточно просто. Для хранения в обоих форматах внутри архива документа используется папка с именем media, как показано на рисунке 20, и соответственно в неё сохраняются все изображения, представленные в контексте документа. В дальнейшем оттуда для отображения изображений данные картинок будут изыматься.



Рисунок 20 – Местоположения изображений в ODT и DOCX документах

В DOCX ссылки на местоположение файлов изображений в архиве хранятся в файле document.xml.rels. В том же файле задаётся идентификационный номер изображения, на который позже будут ссылаться

теги внутри других файлов, в частности в файле document.xml. На рисунке 21 представлен фрагмент файла document.xml.rels.

```
<Relationship Target="media/image1.png" Type="http://schemas.openxmlformats.org/officeDocument/2006/relationships/image" Id="rId5"/>
```

Рисунок 21 – Ссылка на местоположения изображений в DOCX документах

В свою очередь в файле, представляющим весь контент документа, непосредственно внутри тегов, объявляющих параграф и фрагмент документа, создаётся тег <w:drawing> или <w:text> в зависимости от расположения картинки внутри текста или нет. Внутри тега прописываются свойства и параметры, идентификатор изображения. Пример представления изображения представлен на рисунке 22.

```
<w:p w:rsidRDefault="00E51E75" w:rsidR="006D5950" w14:textId="572DDF0C" w14:paraId="03DC7921" w:rsidRPr="001E779D">
  <w:r w:rsidRPr="00E51E75">
    <w:drawing>
      <wp:inline wp14:editId="522707DB" wp14:anchorId="15D2C6F3" distR="0" distL="0" distB="0" distT="0">
        <wp:extent cx="3038899" cy="3067478"/>
        <wp:effectExtent r="9525" b="0" t="0" l="0"/>
        <wp:docPr descr="Изображение выглядит как дно океана Автоматически созданное описание" name="Рисунок 1" id="1"/>
        <wp:cNvGraphicFramePr>
          <a:graphicFrameLocks noChangeAspect="1" xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main"/>
        </wp:cNvGraphicFramePr>
        <a:graphic xmlns:a="http://schemas.openxmlformats.org/drawingml/2006/main">
          <a:graphicData uri="http://schemas.openxmlformats.org/drawingml/2006/picture">
            <pic:pic xmlns:pic="http://schemas.openxmlformats.org/drawingml/2006/picture">
              <pic:nvPicPr>
                <pic:cNvPr descr="Изображение выглядит как дно океана Автоматически созданное описание" name="Рисунок 1" id="1"/>
                <pic:nvPicPr>
                  <pic:blipFill>
                    <a:blip r:embed="rId5"/>
                    <a:stretch>
                      <a:fillRect/>
                    </a:stretch>
                  </pic:blipFill>
                  <pic:spPr>
                    <a:xfrm>
                      <a:off y="0" x="0"/>
                      <a:ext cx="3038899" cy="3067478"/>
                    </a:xfrm>
                    <a:prstGeom prst="rect">
                      <a:avLst/>
                    </a:prstGeom>
                  </pic:spPr>
                </pic:pic>
              </a:graphicData>
            </a:graphic>
          </wp:inline>
        </w:drawing>
      </w:r>
    </w:p>
```

Рисунок 22 – Пример объявления изображений в DOCX документах

В документе ODT хранение рисунков происходит схожим образом. Незначительно отличаются имена тегов: <w:drawing> на <draw:frame>. Но главным отличием является то, что тег напрямую ссылается на местоположение изображения в архиве, как показано на рисунке 23, а не на его идентификационный номер в другом файле.

```
<text:p text:style-name="Обычный">
  <draw:frame draw:name="Рисунок 1" svg:width="3.32338in" draw:style-name="a0" style:rel-height="scale" style:rel-width="scale" svg:height="3.35463in" svg:y="0in" svg:x="0in" text:anchor-type="as-char">
    <draw:image xlink:actuate="onLoad" xlink:show="embed" xlink:type="simple" xlink:href="media/image1.png"/>
    <svg:title/>
    <svg:desc/>
  </draw:frame>
</text:p>
```

Рисунок 23 – Пример объявления изображения в ODT документе

Интересной особенностью DOCX и ODT является то, что существует тег использующиеся для описания изображения. Данное описание может автоматически прописываться после анализа изображения текстовым редактором, например MS WORD.

Сама картинка описывается при помощи n -мерного массива точек, значения которых будут отражать интенсивность того или иного цвета. В случае обычной RGB картинки размера 9×9 поток будет описывать 3 канала составляющие 3 разных цвета: красный, зелёный и синий. Каждый из каналов будет представлять собой таблицу, содержащую 9 колонок и 9 строк что в целом составляет 81 ячейку представляющими собой пиксели на экране. Таким образом поток будет содержать 243 точки, которые при смешивании каналов будут рисовать заданную картинку.

[illegible]

Следует отметить, что данные изображения в документе сжимаются при помощи дискретного косинусного преобразования (DCT).

1.3 Проблемы парсинга PDF документов

Проблема извлечения данных из PDF документа рассмотрена многими авторами, к существующим подходам можно отнести следующие процессы.

Синтаксический анализ на основе графов. Рассмотрен в работе, где предлагается метод анализа текстовых элементов на основе графов путем группировки элементов страницы в соответствии с весами ребер. Так же использование графов позволяет получить хорошие результаты при извлечении текста, шрифта, заголовков, размера межстрочного интервала для абзацев.

Анализ на основе методов машинного обучения. Методы определения таблицы для PDF-документов на основе сверточных нейронных сетей и методов машинного обучения рассмотрены в статье. Так же, на основе машинного обучения предложены варианты извлечения изображений, и контента документа. Однако такие методы из-за возможности ошибки стоит рассматривать скорее как дополняющие, нежели как ключевые при получении структуры и свойств элементов PDF документа.

Математические методы. Методы на основе извлечения и анализа блоков, представлены в статье. А использование модели Маркова для извлечения библиографических данных с библиотекой PDFBox (извлечения информации о тексте и размере шрифта) представлено в статье. Использование различных математических методов и алгоритмов позволяет получить минимальную ошибку при извлечении структурных элементов и их свойств, при минимизации затрачиваемых ресурсов.

Так же, существует достаточно много решений для извлечения формул из PDF документа, основанных на разных алгоритмах, которые могут быть использованы в дальнейшем для развития алгоритмов парсинга PDF документов.

Отдельный интерес представляет работа, где авторы извлекают и классифицируют блоки текста, что может быть полезным на следующем этапе работы при проверке правильности оформления документов формата PDF.

Возможности по извлечению и сегментации текста, различных способах обработки изображений, выделению таблиц рассмотрены авторами работ “Caradoc: a pragmatic approach to PDF parsing and validation» [11], “Extraction and visualization of citation relationships and its attributes for papers in PDF» [12]. Из программных решений были отмечены алгоритмы сегментирования текста в журнальных статьях и выделения цитат в тексте.

Множество программных решений для парсинга PDF позволяют извлекать только текст и изображения. Некоторые из них - отдельные свойства текста. Тем не менее, ни один из вариантов библиотек не подходит для определения корректности оформления структурных элементов документа. Однако существующие решения могут быть использованы для получения свойств и объектов PDF документа:

- 1) PyPDF2, pdfminer, slate на Python выполняют синтаксический анализ. Они позволяют извлекать текст, изображения и метаданные из PDF-файлов.
- 2) PDFBox, iText это библиотеки Java, которые так же могут быть использованы извлечения текста и изображения из PDF-файлов.
- 3) Poppler на C++ позволяет извлекать текст, изображения и метаданные из PDF-файлов и высоко оптимизирован с точки зрения производительности.

В ходе исследования были выделены ключевые особенности и проблемы извлечения информации из PDF документов. Сложность извлечения текстовых абзацев включает в себя два аспекта: технический и особенности формата.

PDF это сложный формат:

- хранит несколько типов информации: текст, графику, изображения и многое другое, что затрудняет последовательное и надежное извлечение информации из файлов PDF,

- PDF-файлы могут быть созданы разными способами из различных источников, включая отсканированные изображения, экспорт из текстовых редакторов и из настольных издательских приложений, каждый из которых создает PDF-файлы по-разному, с разными уровнями структуры и разными методами хранения информации,

- PDF-файлы могут быть зашифрованы, что может затруднить извлечение информации из PDF-файла без ее предварительной расшифровки,

- PDF-файлы могут содержать текст с оптическим распознаванием символов (OCR): некоторые PDF-файлы создаются из отсканированных изображений и содержат текст OCR, который может быть трудно извлечь, что может привести к неточностям или ошибкам,

- PDF-файлы могут содержать нетекстовые элементы: PDF-файлы могут содержать множество нетекстовых элементов, таких как изображения и графика.

В ходе исследования структуры PDF документа были также выделены технические проблемы, которые препятствуют извлечению текстовых абзацев.

Во-первых, PDF хранит текстовые данные посимвольно в отличие от Open Office XML (DOCX) или OpenDocument Text (ODT), где каждый элемент представлен контейнером со стилями, чаще всего в виде абзаца. Таким образом в документах DOCX и ODT появляется возможность извлечь абзац целиком и все характеризующие его стили. В случае с PDF документами это невозможно, что и определяет основную проблему при извлечении элементов документов такого типа и их атрибутов.

Во-вторых, текст в таблицах не имеет каких-либо отличий от обычного текста, так как таблица представляет собой множество графических объектов – прямоугольников, а текст хранится стандартным способом. Таким образом таблица в PDF не имеет единой структуры.

В-третьих, если не соблюдаются правила оформления отчетности, согласно нормативным актам, то определение начала и конца абзацев существенно затрудняется.

В связи с тем, что в основном в PDF содержимое представлено в виде символов и векторной графической информации, было принято решение решать представленные проблемы с точки зрения человеческого восприятия. Другими словами, в основу метода выделения структурных элементов PDF документа была положена аналогия с тем, как, читая отчёт, человек может визуально выделить отдельные абзацы и другие структурные элементы документа. В ходе поиска ответа на заданный вопрос были созданы два алгоритма для выделения строк и параграфов текста, которые будут описаны далее.

1.4 Вывод по главе 1

В ходе поиска и анализа существующих аналогов были выделены их ключевые достоинства и недостатки, изучены используемые ими технологии и ключевые проблемы, являющимися общими для всех решений.

В результате исследования была проанализирована структура PDF, ODT и DOCX форматов текстовых документов. Выделены их различия и особенности в различных категориях сравнения.

В процессе исследования PDF формата были найдены проблемы, которые могут возникнуть при попытке извлечения информации и, в частности, структурных элементов текстовых документов и их свойств.

Во 2 главе предложены способы решения описанных проблем в виде спроектированных алгоритмов.

2. Алгоритмы для извлечения структурных элементов

Система парсинга данных из PDF состоит из нескольких этапов.

Первый этап заключается в выделении всей хранимой в документе информации, которая включает в себя: каталог страниц, символы, их позиционирование на странице, кегль, шрифты, рисунки, различные нарисованные объекты и т. п.

Второй этап подразделяется на два подэтапа выполняемых параллельно:

- объединение символов на основании их местоположения в строки,
- извлечение рисунков,
- извлечение таблиц и их текста из всех страниц документа.

Третий этап заключается в удалении строк текста, дублирующих содержимое таблиц, из списка.

Четвертый этап выполняет расчёт межстрочного интервала между всеми строками, в случае если 2 строки находятся на разных страницах, то его значение приравнивается к нулю.

Все предыдущие этапы были необходимы для сбора и вычисления информации, которая нужна для выделения абзацев текста из списка строк.

Используя собранную информацию на последнем этапе, выделяются текстовые абзацы.

В целом этапы работы системы представлены на рисунке 25.

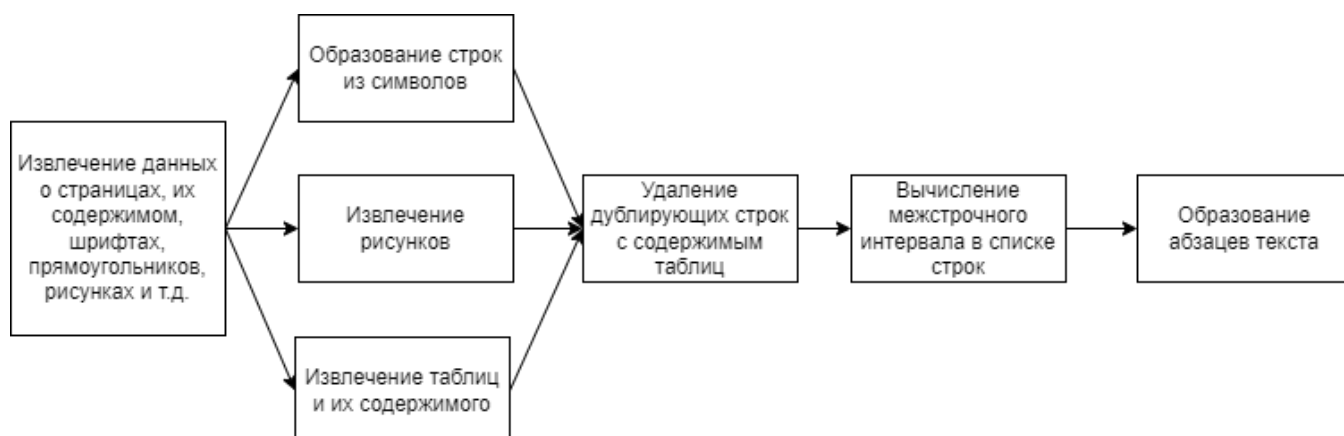


Рисунок 25 – Этапы работы системы извлечения абзацев, их атрибутов и свойств

В ходе исследований использовалось множество инструментов и технологий для извлечения первичных данных pdf документа.

Для парсинга первичных данных в основном использовалась библиотека pdfplumbert, которая имеет следующий функционал необходимый для проведения исследования:

- извлечение объектов pdf документа, которые включают в себя:
 - a) символы;
 - b) шрифты;
 - c) позиции символов;
 - d) рисунки;
 - e) графические объекты;
- извлечение полноценных таблиц в виде отдельного объекта класса;
- извлечение метаданных, таких как:
 - a) родительское приложение;
 - b) данные об авторе;
 - c) время создания и модернизации файла;
- данные о размере страниц;
- визуальная отладка.

2.1 Алгоритмы для строк

В основе алгоритма формирования списка строк лежит информация о позиции каждого символа на странице. По всем символам страницы проходит цикл и вычисляет, является ли позиция текущего символа по оси ординат такой же, как и у предыдущего символа. В случае если позиции одинаковы, то символ добавляется в строку, если разные, то строка добавляется в список, а текущий символ начинает новую строку.

Система координат PDF документа начинается с нижнего левого края страницы. Таким образом, чем меньше значение оси ординат, тем ниже и соответственно левее на странице расположен объект.

Алгоритм образования строк полностью представлен на блок-схеме на рисунке 26.

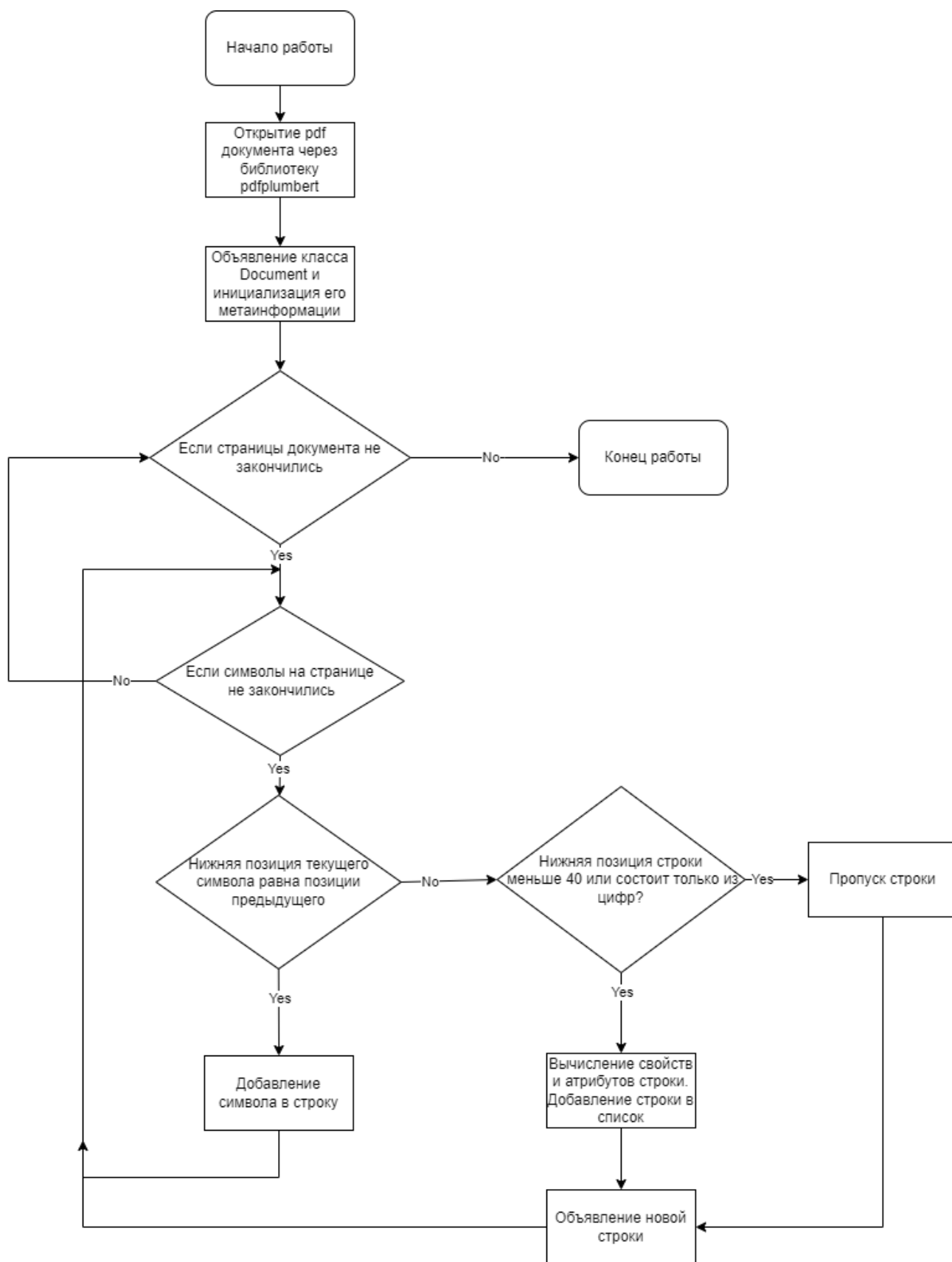


Рисунок 26 – Блок-схема алгоритма образования строк

Так как отчеты могут содержать нумерацию и колонтитулы, которые в спецификации PDF идут на первом месте страницы в порядке следования содержимого, то в таких случаях они мешают вычислению данных, необходимых для последующей работы системы. Для решения этой проблемы используется порог отступа по оси у равный сорока топографическим пунктам, а также дополнительно проверяется не состоит ли строка только из цифр. В случае если условие выполняется, то данная строка не добавляется в список и пропускается.

2.2 Алгоритмы для извлечения текстовых параграфов

Как уже говорилось ранее разработка алгоритма для выделения текстовых абзацев вдохновлялась человеческим восприятием параграфов в текстовых документах. Мной было выделено несколько факторов по которым человек ориентируется при данном выборе:

- различия в межстрочном интервале между абзацами,
- различия в отступе от красной строки, так как известно, что первая строка нового абзаца должна начинаться с отступа от начала страницы,
- последняя строка абзаца очень часто не доходит до конца страницы.

Основываясь на данных факторах, был разработан алгоритм, показанный на блок-схеме на рисунке 27.

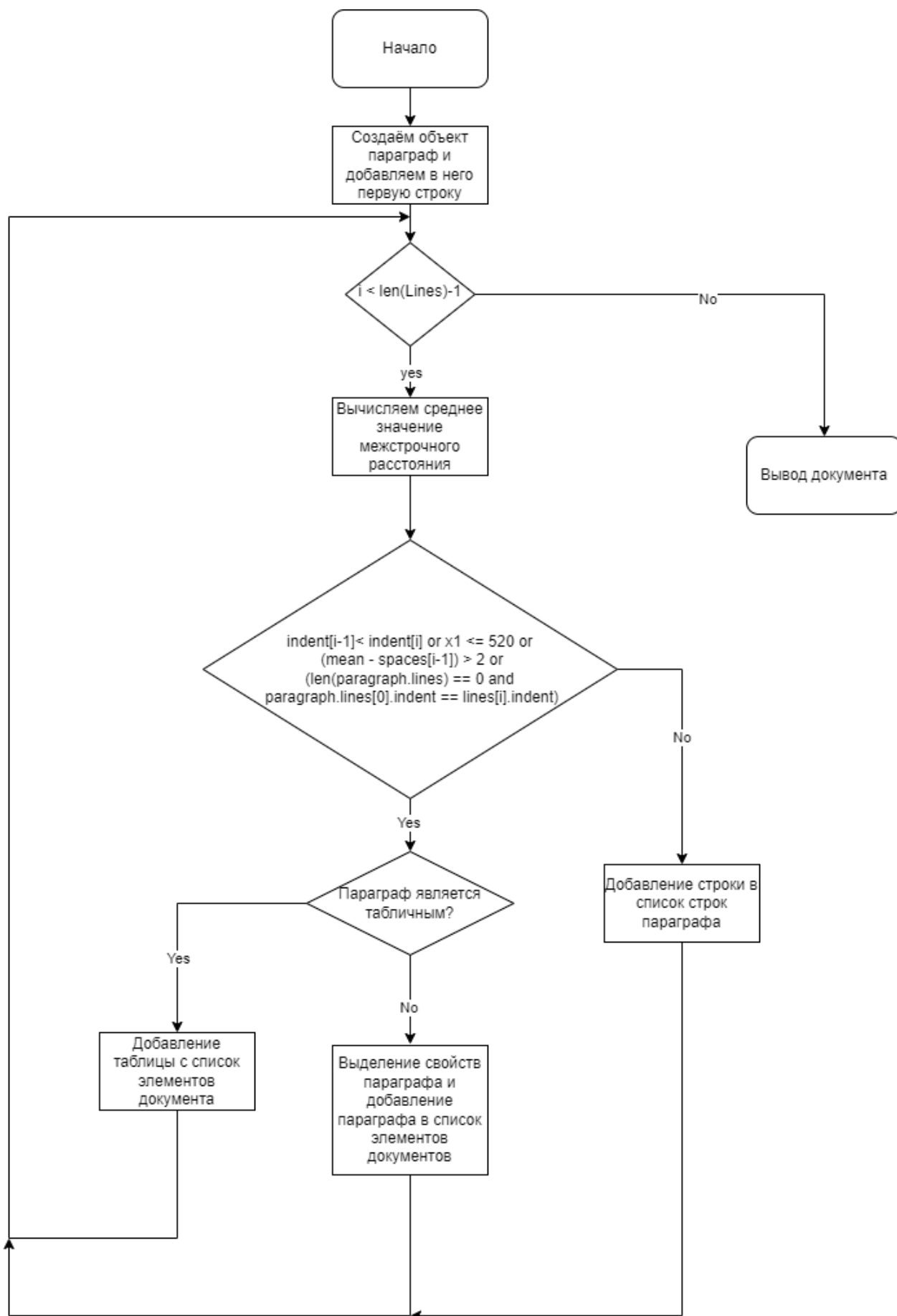


Рисунок 27. Блок-схема образования параграфов

Блок схема имеет следующие обозначения:

- lines - список строк,
- spaces - список межстрочных интервалов относительно списка строк,
- mean - средний межстрочный интервал в параграфе;
- paragraph - объект абзаца,
- indent - отступ от красной строки.

Главными условиями выделения параграфов в алгоритме являются различия в отступах от левой и правой граней страницы и межстрочный интервал двух строк. Другими словами, если наблюдается сильное различие, то алгоритм понимает, что начинается новый абзац.

Для лучшего восприятия алгоритма нами представлен рисунок 28.

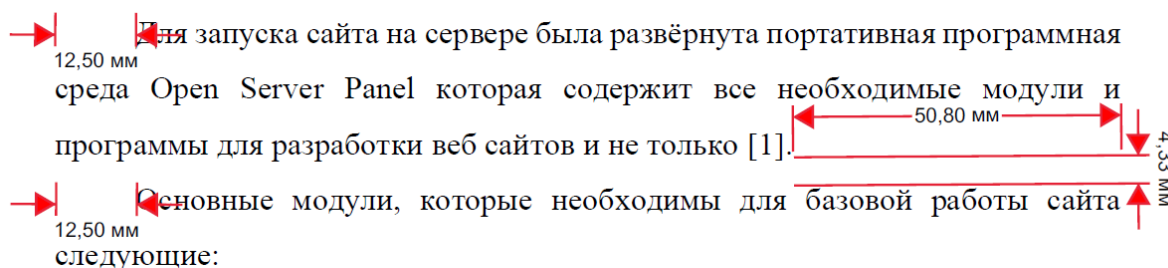


Рисунок 28. Визуальный пример определения параграфов

В ходе исследования нами был выбран оптимальный порог отступа крайней строки параграфа от правого края страницы А4 равный 520 топографических пунктов. Данный порог можно корректировать в зависимости от значений полей страниц документа.

В связи с тем, что некоторые строки могут иметь заглавные символы, а некоторые нет, то при проверке различий в межстрочном интервале была введена погрешность в 2 пункта.

2.3 Алгоритмы для извлечения свойств текстовых параграфов

Для выполнения нормоконтроля документации необходимо знать значения свойств и атрибутов абзацев. Для их выделения был разработан следующий, представленный на рисунке 29 алгоритм.

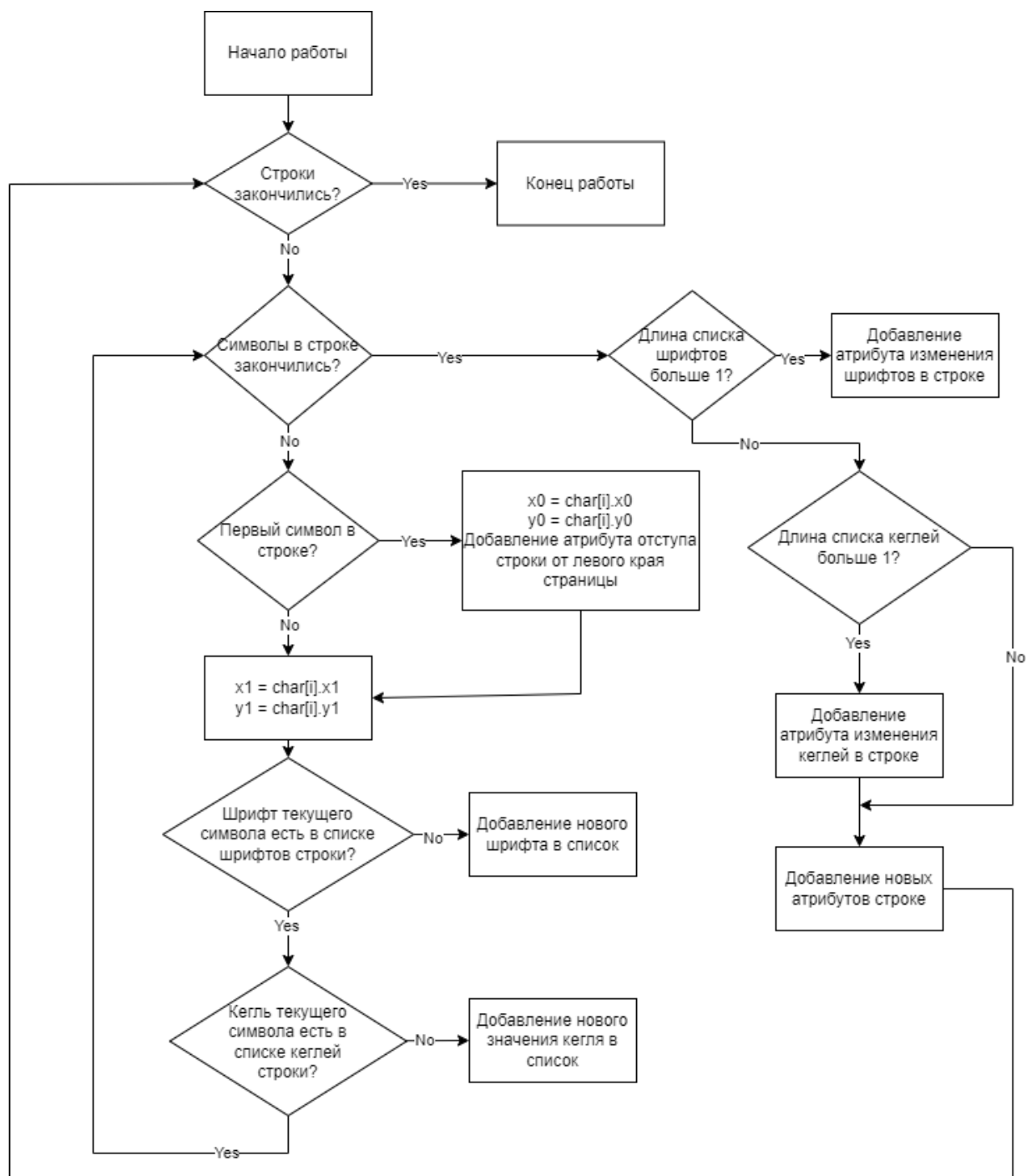


Рисунок 29 – алгоритм образования атрибутов строк и абзацев

Алгоритм работает следующим образом. При образовании строк из символов сохраняются данные о расположении первого и последнего символа, на которых формируется данные о расположении всей строки. Для понимания изменялся ли шрифт или кегель при переходе на следующий символ строки выполняется поиск его шрифта и кегля в соответственных списках. В случае если такого шрифта или кегля нет, то он добавляется в список. В результате

обработки всех символов строки, если длина соответствующего списка больше единицы, то к нему добавляется атрибут, оповещающий о неоднородности шрифта или кегля.

Аналогичным образом происходит формирование атрибутов абзацев, только поиск ведётся уже в списке строк. В дополнение к шрифту, кеглю и местоположению в тексте добавляется значение красной строки.

2.4 Алгоритмы для извлечения рисунков и таблиц

Для извлечения таблиц и рисунков в системе используется библиотека `pdfplumber`. Они извлекаются постранично и в результате формируют списки таблиц и рисунков соответственно.

В ходе исследования обнаружилось что в некоторых случаях технологии обнаружения табличных объектов библиотеки выделяют таблицы в тех местах, где их нет. В результате более детального анализа первичных выделяемых данных выяснилось, что в некоторых местах на странице могут существовать объекты линий и прямоугольников, по которым и выделяются таблицы. Одной из возможных причин может являться ошибка при конвертации в PDF формат документов других форматов.

Для решения данной проблемы было принято решение использовать технологию извлечения таблиц другой библиотеки – `tabula` для подтверждения действительности существования таблицы. Это уменьшило скорость работы парсера, но увеличило точность выделения таблиц. Алгоритм извлечения таблиц из документа представлен на рисунке 30.

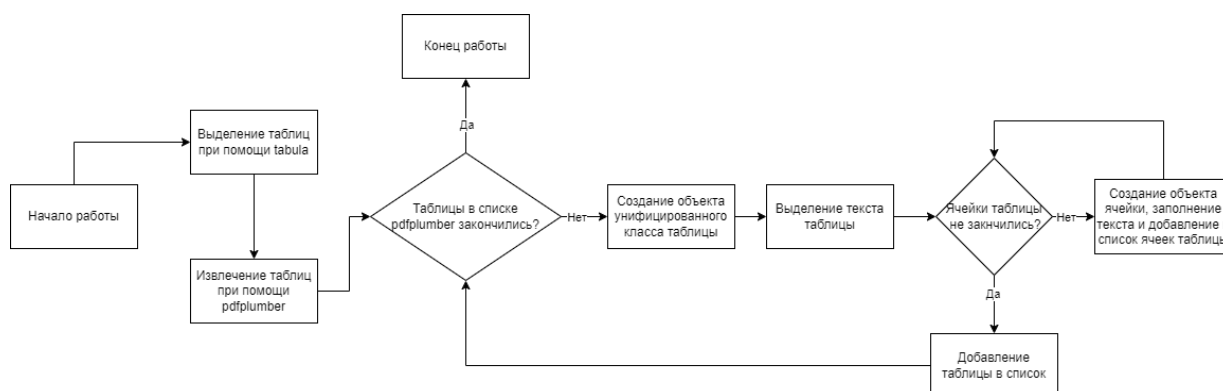


Рисунок 30 – Алгоритм образования атрибутов строк и абзацев

Таблицы имеют атрибуты расположения, количество столбцов и строк. Также таблицы имеют список текстовых строк по ячейкам. В ходе работы описанных ранее этапов дубликаты абзацев и текстовых строк таблиц удаляются, оставляя вместо себя только объект таблицы.

Рисунки также имеют атрибуты месторасположения на странице, размера и используемого формата. В зависимости от расположения рисунка в документе, объект моделирующий рисунок добавляется в список элементов.

2.5 Тестирование работы алгоритмов

Система была протестирована на 30 отчетах по 25–34 страницы по научно-исследовательским работам магистрантов.

Тестирование проводилось посредством сохранения извлеченных абзацев в CSV файл и последующим ручным сравнением трёх экспертов.

Пример образованного csv файла представлен на рисунке 31.

| | A | B | C | D | E | F | G | H |
|----|---|--------------|----------|-----------|-----------|---------|-------------|--------|
| 1 | text | countOfSpSbl | countSbl | uppercase | lowercase | lastSbl | firstkey | indent |
| 2 | Введение | 0 | 9 | ЛОЖЬ | ЛОЖЬ | e | TitleLevel1 | 7.88 |
| 3 | В ходе предыдущей исследовательской работы было выявлено несколько ключевых пр | 1 | 137 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 4 | Среди них мной были выбраны по моему мнению наиболее актуальные и срочные для | 2 | 107 | ЛОЖЬ | ЛОЖЬ | : | TitleLevel1 | 1.25 |
| 5 | – создание базы данных приложения, включающее в себя схему хранения правил соотв | 2 | 132 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.25 |
| 6 | – оптимизация работы процесса обмена информацией между клиентской частью и серв | 1 | 85 | ЛОЖЬ | ИСТИНА | ; | listLevel1 | 1.25 |
| 7 | – оптимизация и адаптация существующей модели парсинга текстовых документов к раб | 3 | 113 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.25 |
| 8 | – исследование возможности улучшения и оптимизации существующего классификатор | 1 | 103 | ЛОЖЬ | ИСТИНА | . | listLevel1 | 1.25 |
| 9 | Цель работы заключается в решение поставленных на предыдущем этапе ключевых пр | 1 | 118 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 10 | Задачи работы: | 1 | 15 | ЛОЖЬ | ЛОЖЬ | : | TitleLevel1 | 1.25 |
| 11 | – развернуть серверную составляющую сервиса; | 1 | 45 | ЛОЖЬ | ИСТИНА | ; | listLevel1 | 1.25 |
| 12 | – спроектировать и реализовать модель база данных; | 1 | 51 | ЛОЖЬ | ИСТИНА | ; | listLevel1 | 1.25 |
| 13 | – оптимизировать процесс взаимодействия клиента и сервера; | 1 | 59 | ЛОЖЬ | ИСТИНА | ; | listLevel1 | 1.25 |
| 14 | – оптимизировать структуру модуля парсинга данных; | 1 | 51 | ЛОЖЬ | ИСТИНА | ; | listLevel1 | 1.25 |
| 15 | – оптимизировать модель классификатора параграфов. | 1 | 52 | ЛОЖЬ | ИСТИНА | . | listLevel1 | 1.25 |
| 16 | 1 Подковка аппаратных и программных средств, необходимых для разработки, тестиро | 3 | 136 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 17 | В ходе предыдущих исследований было решено разделить работу сервиса на две части: | 5 | 272 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 18 | Основной причиной этого стало то, что большинство пользователей MS Word никогда н | 7 | 378 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 19 | Поэтому было решено переделать клиентскую часть в виде веб-сайта, к которому у люд | 6 | 306 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 20 | Для начала полноценной работы был развернут VPS сервер со следующими характерис | 1 | 87 | ЛОЖЬ | ЛОЖЬ | : | TitleLevel1 | 1.25 |
| 21 | – операционная система: Windows Server 2019; | 2 | 45 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.89 |
| 22 | – процессор: Intel Xeon, 2x2.2ГГц; | 4 | 35 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.89 |
| 23 | – оперативная память: 2 Гб; | 2 | 28 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.89 |
| 24 | – жесткий диск: HDD 40 Гб в RAID 1. | 2 | 36 | ЛОЖЬ | ЛОЖЬ | . | listLevel1 | 1.89 |
| 25 | Серверу был выделен IPv4-адрес 195.133.197.161, который служит для постоянного подк | 7 | 162 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 26 | Для удобства будущих пользователей был куплен домен normcontrol.ru и normcontrol. | 6 | 227 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 27 | Рисунок 1 – Записи домена normcontrol.ru | 1 | 41 | ЛОЖЬ | ЛОЖЬ | u | TitleLevel1 | 3.83 |
| 28 | Для запуска сайта на сервере была развернута портативная программная среда Open Se | 1 | 188 | ЛОЖЬ | ЛОЖЬ | . | TitleLevel1 | 1.25 |
| 29 | Основные модули, которые необходимы для базовой работы сайта следующие: | 2 | 72 | ЛОЖЬ | ЛОЖЬ | : | TitleLevel1 | 1.25 |
| 30 | – Apache: 2.2.31, 2.4.38, 2.4.41, 2.4.53 + auth_ntlm, fcgid, maxminddb, xsendfile; | 16 | 83 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.25 |
| 31 | – Bind: 9.16.28; | 4 | 17 | ЛОЖЬ | ЛОЖЬ | ; | listLevel1 | 1.25 |

Рисунок 31 – Пример образованного CSV файла

В ходе тестирования было обнаружено 257 ошибок определения параграфа на 4338 параграфов, как показано на рисунке 32.

Статистика ошибок

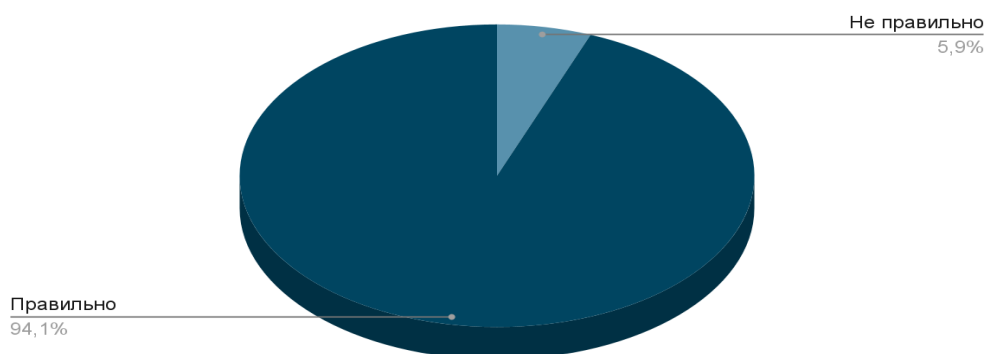


Рисунок 32 – Диаграмма процента ошибок парсинга

Примерно 81% ошибок произошло по причине ошибочного составления строк. Основной причиной данных ошибок является различное оформление студентами отступов от левой границы страницы.

Было найдено два случая (0,8%), где находились случайные переходы на новый абзац в середине предложения.

Алгоритм не смог правильно обработать подписи к рисункам, состоящие из нескольких строк в связи с выравниванием по центру, что составило примерно 7,4% от всех ошибок.

Также были обнаружены ошибки при переходе наименования рисунка к обычному абзацу, т. к. позиции строк совпали.

Диаграмму распределения ошибок по типам представлена на рисунке 33.

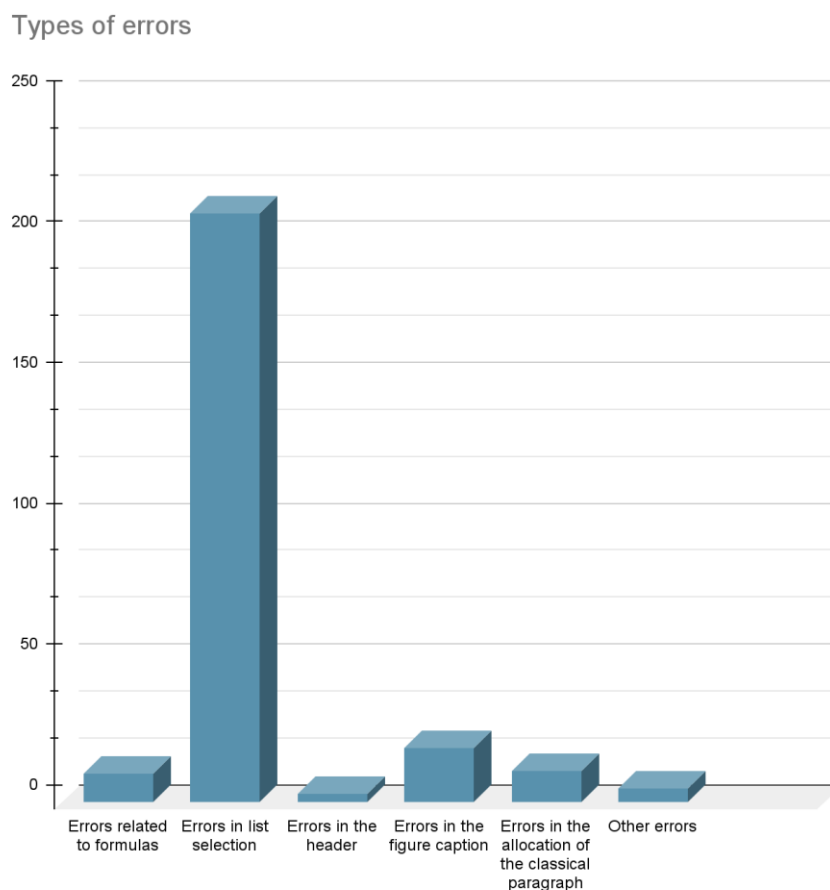


Рисунок 33 – Диаграмма распределения ошибок по типу

2.5 Вывод по главе 2

В ходе разработки алгоритмов парсинга PDF документов были решены следующие проблемы:

- отсутствие в структуре PDF формата объектов-контейнеров, которые содержат в себе данные о текстовых параграфах,
- отсутствие объектов-таблиц и рисунков,
- отсутствие некоторых необходимых для нормоконтроля свойств структурных элементов.

В результате проведённой работы были получены алгоритмы для:

- образования строк из символов,
- образования параграфов из строк,
- образования новых свойств из уже имеющихся,
- выделения таблиц и рисунков.

В связи с тем, что алгоритмы парсинга будут использоваться в сервисе, который предполагает поддержку нескольких форматов файлов, некоторые его модули необходимо адаптировать и модернизировать для работы с PDF, ODT и другими форматами. Процесс адаптации и модернизации описан в 3 главе данной работы.

3. Модификация веб-сервиса

3.1 Унифицированные классы структурных элементов

В проекте предполагается использовать технологии для парсинга трёх различных форматов документов: PDF, ODT, DOCX. Однако внутренняя структура этих документов сильно различается.

В то же время следует учитывать, что все текстовые документы имеют одинаковые структурные элементы с точки зрения содержания. Например, такие элементы как таблица, изображение или обычный текстовый параграф всегда присутствуют в ВКР или диссертациях.

Для решение данной проблемы были разработаны специальные унифицированные классы, представляющие структурные элементы отчёта.

На таблице 3 представлены все типы структурных элементов и их представление в проекте.

Таблица 3 – Типы структурных элементов

| Наименование структурного элемента | Наименование унифицированного класса |
|------------------------------------|--------------------------------------|
| Документ | UnifiedDocumentView |
| Текстовый абзац | Paragraph |
| Список | List |
| Таблица | Table |
| Рамка | Frame |
| Формула | Formules |
| Изображение | Image |

На рисунке 34 представлена диаграмма классов по нотации UML, на которой показаны все реализованные унифицированные классы.

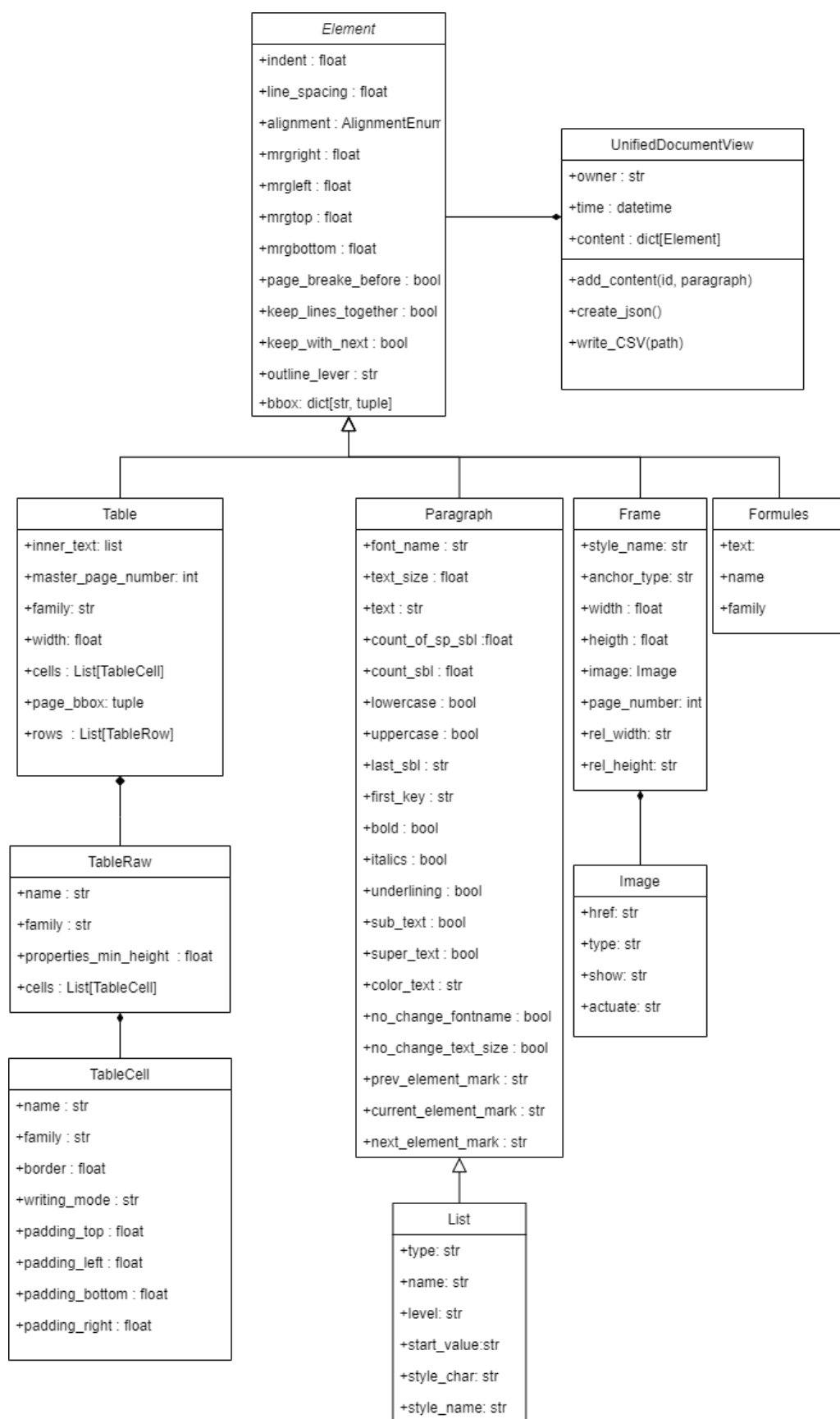


Рисунок 34 – Диаграмма классов унифицированных представлений
структурных элементов

Первоначально был выделен родительский класс `Element`, который содержит атрибуты присущие каждому элементу. К таким атрибутам относятся: отступы от граней страницы, выравнивание по горизонтали, отступ от красной строки, межстрочный интервал, и некоторые другие показатели, описанные в таблице 4.

Таблица 4 – Атрибуты родительского класса `Element`

| Название признака | Тип | Свойство |
|---------------------------------|----------------------|--|
| <code>alignment</code> | <code>String</code> | Выравнивание |
| <code>indent</code> | <code>Double</code> | Красная строка |
| <code>mrgrg</code> | <code>Double</code> | Отступ справа |
| <code>mrglf</code> | <code>Double</code> | Отступ слева |
| <code>lineSpacing</code> | <code>Double</code> | Межстрочный интервал |
| <code>mrgtop</code> | <code>Double</code> | Отступ сверху |
| <code>mrgbtm</code> | <code>Double</code> | Отступ снизу |
| <code>keep_together</code> | <code>Boolean</code> | Не разрывать абзац |
| <code>keep_with_next</code> | <code>Boolean</code> | Не отрывать от следующего |
| <code>page_breake_before</code> | <code>Boolean</code> | С новой страницы |
| <code>outline_level</code> | <code>String</code> | Тип элемента со стороны текстового редактора |
| <code>bbox</code> | <code>Dict</code> | Позиция структурного элемента на странице |

Далее были образованы классы представляющие структурные элементы и наследующие класс `Element`. Для них были выделены присущие только им атрибуты.

Класс `Paragraph` содержит классические свойства, относящиеся к текстовому абзацу. Такие атрибуты напрямую необходимы в проверке на

соответствие ГОСТу и к ним чаще всего предъявляются требования в нормативных актах. Классические атрибуты описаны в таблице 5.

Таблица 5 – Классические атрибуты параграфа

| Название признака | Тип | Свойство |
|-------------------|---------|----------------------|
| text | String | Текст параграфа |
| bold | Boolean | Жирность |
| italics | Boolean | Курсив |
| underlining | Boolean | Подчёркивание текста |
| sub_text | Boolean | Подтекст |
| super_text | Boolean | Надтекст |
| color_text | List | Цвета текста |
| font_name | List | Шрифты текста |
| text_size | List | Размеры текста |

Кроме того, в связи с тем, что класс параграфа на дальнейших этапах проходит процесс классификации были выделены и в последствии добавлены необходимые для работы классификатора данные. Они были названы производными так как вычисляются из классических атрибутов. Например, на основании данных о тексте вычисляются количество символов, первый и последний символ. Производные атрибуты представлены в таблице.

Таблица 6 – Производные атрибуты

| Название признака | Тип | Свойство |
|-------------------|---------|---------------------------------|
| PrevElementMark | String | Класс предыдущего элемента |
| CurElementMark | String | Класс текущего элемента |
| NextElementMark | String | Класс следующего элемента |
| count_of_sp_sbl | Integer | Количество специальных символов |
| count_sbl | Integer | Количество всех символов |
| lowercase | Boolean | Весь текст с нижнего регистра |

| | | |
|-----------|---------|--------------------------------|
| uppercase | Boolean | Весь текст с верхнего регистра |
| last_sbl | String | Последний символ |
| first_key | String | Первый символ |

Для сохранения извлечённых данных документа был разработан класс `UnifiedDocumentView`, который позволяет хранить метаданные и все данные о содержании документа в виде последовательности структурных элементов.

Кроме этого, в класс, был добавлен метод сохранения полученного результата в виде csv файла. В основе метода лежит функционал библиотеки `dataclasses`. Программный код, реализующий данную функцию представлен на рисунке 35.

```
try:
    if len(self.content) < 1:
        raise DocumentEmptyContentException
    content = {}
    json_text = {'owner': self.owner, 'time': self.time, 'page_count': self.page_count}
    for key, values in self.content.items():
        temp = {'element_class': type(values).__name__}
        for attribute_name, value in dataclasses.asdict(values).items():
            temp[attribute_name[1::]] = value
        content[key] = temp
    json_text['content'] = content
    json_text = json.loads(json.dumps(json_text))
    return json_text
except DocumentEmptyContentException as e:
    print(e)
    raise
```

Рисунок 35 – Функция образования JSON строки

Также при разработке системы предполагалось, что она будет поддерживать несколько различных форматов текстовых документов. Однако они могут иметь разные единицы измерения, например, `odt` измеряет длину в дюймах, а `pdf` в топографических пунктах. Для решения данной проблемы были добавлены функции конвертации дюймов в см. и топографических пункты в см. и в обратную сторону.

3.2 Классификатор текстовых структурных элементов документа

Для проверки текстовых документов и исправления обнаруженных ошибок необходимо первоначально определить к каким классам принадлежат параграфы. В зависимости от класса параграфа по ГОСТу применяются различные правила оформления. Например, в основном весь текст выпускной квалификационной работы должен иметь выравнивание по ширине, однако для заголовка “Введение” и “Выводы” применяется выравнивание по центру. Таким образом для правильной работы сервиса необходимо провести классификацию параграфов документа.

Классификатор применяется сервисом только в трёх случаях:

- если из текущего формата документа не удалось извлечь данные о типах параграфов,
- при обработке PDF документа,
- при неопределённости типа структурного элемента.

Классификатор использует алгоритм градиентного бустинга, основанный на открытой технологии CatBoost, разработанной компанией «Яндекс». Основные параметры алгоритма были выбраны посредством множественных тестовых запусков. Основными критериями выбора являлись: эффективность классификатора и время его работы.

3.2.1 Описание существующего решения

В ходе исследования была проанализирована работа алгоритма классификации схожего проекта, разрабатываемого в 2019–2021 года в рамках НИРМА в университете ИТМО.

Основой модели классификации проекта служила технология Catboost.

В качестве датасета использовались выборки выпускных работ бакалавров и магистров. Из них были удалены все персональные данные, такие как имя автора, имя ПК и т. п. В выборке присутствовала информация только об основном теле документа, остальные данные такие как список использованных источников, титульный лист, реферат были удалены. Фрагмент датасета представлен на рисунке 36.

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z |
|----|------------|-----------|----------|----------|-----------|-----------|-----------|-----------|------------|---------|----------|----------|------------|----------|-----------|----------|------------|-----------|----------|-----------|----------|-----------|----------|----------|--------------|--------|
| 1 | Content | SpecialSy | WordsCoc | SymbolCo | lowercase | uppercase | LastSymbi | FirstKeyW | PrevElem | CurElem | NextElem | FullBold | FullItalic | Alignmen | KeepLines | KeepWith | LeftFinden | LineSpaci | NoSpaceB | OutlineLe | PageBrea | RightInde | SpaceAft | SpaceBef | SpecialInden | |
| 2 | Введение | 0 | 1 | 8 | False | False | . | | c0 | b1 | c0 | c0 | True | False | Justify | False | True | 0,15 | False | Level1 | False | 0 | 0 | 0 | 0 | -35,45 |
| 3 | Сервисы | 9 | 48 | 386 | False | False | . | | b1 | c0 | c0 | c0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 4 | Эту задачу | 12 | 67 | 535 | False | False | . | | c0 | c0 | c0 | c0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 5 | Косвенно | 11 | 54 | 405 | False | False | . | | c0 | c0 | c0 | c0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 6 | Побужден | 1 | 5 | 37 | False | False | . | | c0 | c0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 7 | Результат | 4 | 30 | 275 | False | False | . | | d0 | c0 | c0 | c0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 8 | Полонен | 2 | 20 | 158 | False | False | . | | c0 | c0 | c0 | c0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 9 | Исходя из | 8 | 33 | 286 | False | False | . | | c0 | c0 | c0 | c0 | False | False | Justify | False | True | 0,15 | False | Level2 | False | 0 | 0 | 0 | 0 | -35,45 |
| 10 | Для обес | 1 | 8 | 69 | False | False | . | | c0 | c0 | d0 | d0 | True | False | Justify | False | True | 0,15 | False | Level2 | False | 0 | 0 | 0 | 0 | -35,45 |
| 11 | - изучить | 2 | 4 | 30 | True | False | . | | listLevel1 | c0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 12 | - выбрать | 2 | 4 | 30 | True | False | . | | listLevel1 | d0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 13 | - формал | 3 | 8 | 60 | True | False | . | | listLevel1 | d0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 14 | - формул | 3 | 9 | 66 | True | False | . | | listLevel1 | d0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 15 | - реализ | 2 | 3 | 24 | True | False | . | | listLevel1 | d0 | d0 | d0 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 16 | В ИМЯ? | 2 | 3 | 22 | True | False | . | | listLevel1 | d0 | d0 | b1 | False | False | Justify | False | False | 0,15 | False | BodyText | False | 0 | 0 | 0 | 0 | -35,45 |
| 17 | 1 ОПИСА | 3 | 11 | 82 | False | True | . | | TitleLevel | d0 | b1 | b23 | True | False | Center | False | True | 0,15 | False | Level1 | False | 0 | 0 | 0 | 0 | -35,45 |
| 18 | 1.1 Элект | 4 | 5 | 55 | False | False | . | | TitleLevel | b1 | b23 | c0 | False | False | Justify | False | False | 53,45 | 1,5 | True | BodyText | False | 0 | 10 | 0 | -35,45 |
| 19 | Электрон | 9 | 34 | 279 | False | False | . | | b23 | c0 | c0 | c0 | False | False | Justify | False | False | 5,25 | 1,5 | False | BodyText | False | 0 | 0 | 0 | -35,45 |

Рисунок 36 – Фрагмент исходного датасета

Были определены свойства, определяющие оформление элементов классов, на основе которых была произведена разметка. Разметка была сделана в два этапа: сначала по более общей группе (абзац, список, заголовок) и после по более мелким группам, то есть различным видам абзацев, заголовков, списков. Например, заголовки были поделены на заголовки первого уровня и заголовки уровней 2–3; элементы списка поделены на первый, последний и средний. Подобные разделения предположительно должны были дать улучшение предсказаний модели. В результате первоначальный набор данных для классификатора содержал 14 классов. Полный список классов элементов и описание каждого из них представлены в таблице 7.

Таблица 7 – Классы элементов

| Наименование класса | Описание |
|---------------------|-------------------------------|
| b1 | Заголовок 1 уровня |
| b2 | Заголовок 2-3 уровней |
| c1 | Обычный абзац |
| c2 | Абзац перед списком |
| d2 | Элемент перечисления (списка) |
| f1 | Подпись к таблице |
| f2 | Таблица |
| f3 | Продолжение таблицы |
| g1 | Рисунок |
| h1 | Подпись к рисунку |

| | |
|----|-------------------|
| i1 | Формула |
| j0 | Подпись к формуле |

Для наглядности на рисунке 37 представлена диаграмма, визуализирующая количество параграфов каждого класса во всём датасете.

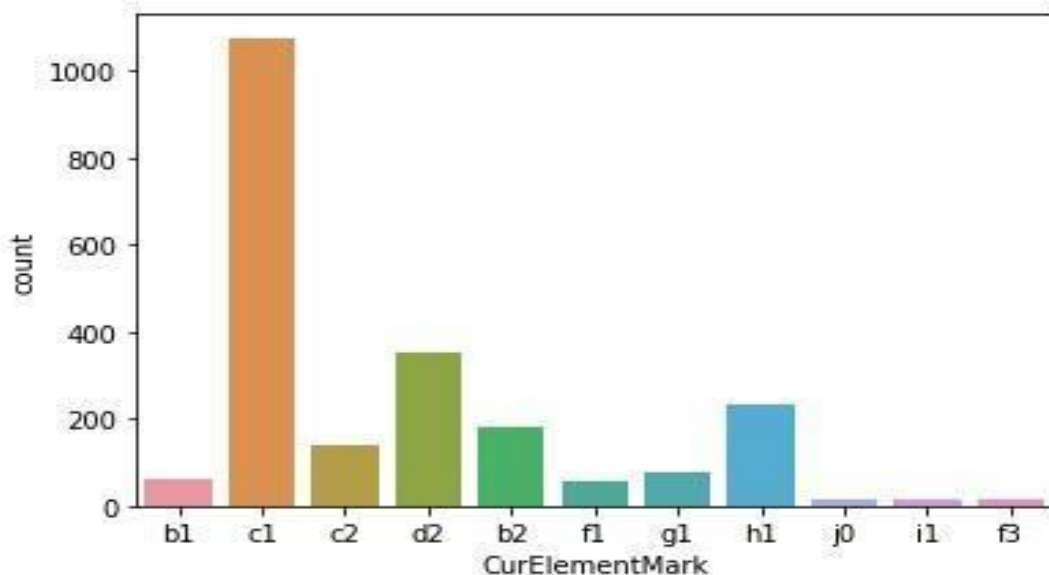


Рисунок 37 – Диаграмма распределения классов датасета

Для обучения алгоритма правильный класс элемента был занесён в столбец CurElementMark. В ходе исследований исследователями было выявлено что точность предсказания класса элемента сильно зависит от класса предыдущего и следующего параграфа, которые были занесены соответственно в столбцы PrevElementMark и NextElementMark.

Остальные свойства были собраны при помощи модуля парсинга docx документов DocxCorrector посредством обработки порядка 10 текстовых документов магистерских и бакалаврских выпускных квалификационных работ. Все входные признаки представлены в таблице 8.

Таблица 8 – Признаки, участвующие в обучении классификатора

| Название признака датасета | Описание |
|----------------------------|------------------------------|
| SpecialSymbolsCount | Количество знаков препинания |
| WordsCount | Количество слов элемента |
| SymbolCount | Количество символов |

| | |
|--------------------------------------|---|
| Lowercase | Весь текст нижним регистром |
| SymbolCount | Количество символов |
| Lowercase | Весь текст нижним регистром |
| Uppercase | Весь текст верхним регистром |
| LastSymbolPd | Символ в конце элемента |
| FirstKeyWord | Ключевое слово класса |
| PrevElementMark | Класс предыдущего элемента |
| CurElementMark | Класс текущего элемента |
| NextElementMark | Класс следующего элемента |
| FullBold | Жирный шрифт элемента |
| FullItalic | Курсив |
| Alignment | Расположение элемента |
| KeepLinesTogether | Между строками нет разрывов или таблиц |
| KeepWithNext | Не отрывать элемент от следующего |
| LeftIndentation | Отступ слева |
| LineSpacing | Междустрочный интервал |
| NoSpaceBetweenParagraphsOfSame Style | Расстояние между параграфами одного стиля |
| OutlineLevel | Уровень заголовка |
| PageBreakBefore | Разрыв страницы перед Документом |
| RightIndentation | Отступ справа |
| SpaceAfter | Отступ после |
| SpaceBefore | Отступ перед |
| SpecialIndentation | Отступ первой строки |

Для оценки оптимизации модели классификатора использовались следующие метрики качества:

- accuracy Score,

- precision,
- recall,
- f1-score,
- macro Average,
- weighted Average,
- confusion Matrix.

Accuracy Score показывает сколько классов были определены правильно относительно общего количества прогнозов. Для определения остальных метрик использовалась библиотека Sklearn – classification_report. Она позволяет определить не только такие метрики как, Macro F1 и Weighted F1, но и выделить метрики Precision, Recall, F1-score и Support относительно каждого класса.

Как видно из рисунка диаграммы распределения классов, количество классов в датасете не сбалансировано, поэтому был применён алгоритм RandomOverSampler, который в зависимости от настроек дублирует определённые строки набора данных.

При обучении и тестировании алгоритма получились следующие данные, представленные в таблице 9.

Таблица 9 – Результаты тестирования классификатора

| Номер класса | Precision | Recall | F1-score | Support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 0.62 | 0.77 | 58 |
| 1 | 0.86 | 0.92 | 0.88 | 155 |
| 2 | 0.92 | 0.94 | 0.93 | 908 |
| 3 | 0.65 | 0.63 | 0.64 | 120 |
| 4 | 0.93 | 0.91 | 0.92 | 297 |
| 5 | 1.00 | 1.00 | 1.00 | 42 |
| 6 | 1.00 | 1.00 | 1.00 | 12 |
| 7 | 0.98 | 0.98 | 0.98 | 66 |
| 8 | 0.99 | 1.00 | 1.00 | 190 |

| | | | | |
|--------------|------|------|------|------|
| 9 | 1.00 | 1.00 | 1.00 | 923 |
| 10 | 1.00 | 1.00 | 1.00 | 922 |
| Accuracy | | | 0.96 | 3693 |
| Macro avg | 0.94 | 0.91 | 0.92 | 3693 |
| Weighted avg | 0.96 | 0.96 | 0.96 | 3693 |

3.2.2 Проектирование и оценка модели классификации текстовых структурных элементов документов

В ходе исследования работы описанного ранее классификатора было выделено несколько ключевых аспектов и проблем.

Первой среди выделенных проблем была слишком большая размерность пространства признаков. При исследовании было выявлено что это количество может быть очень сильно уменьшено. Также за счёт большего количества параметров проходящий трафик между клиентом и сервером может быть достаточно большим.

Второй и основной проблемой является неправильно поставленная на предыдущих этапах перед классификатором задача. При подготовке датасета были использованы уже сданные и прошедшие нормоконтроль отчёты. Таким образом полученные данные были пригодны для использования классификации правильно оформленных параграфов, возможно с небольшими ошибками. Однако главной задачей классификатора является классификация параграфов независимо от его оформления в документе. Также стоит учитывать, что количество вариантов ошибок в оформлении элементов может стремиться к бесконечности, что делает представленную задачу очень сложной.

В качестве возможных решений можно было использовать два варианта:

- подготовка нового датасета из ещё не прошедших нормоконтроль отчётов,

– изменить имеющийся датасет таким образом, чтобы он был минимально зависим от оформления элементов.

Как уже упоминалось ранее количество ошибок оформления может быть очень велико, также следует учитывать, что собрать такой датасет может быть сложной задачей, т. к. такие документы являются интеллектуальной собственностью человека, а до официального оформления, в т. ч. нормоконтроля авторы будут негативно относиться к таким процессам. Поэтому первоначально предполагалось использовать второй вариант, но в ходе более детального исследования было выявлено, что используемый датасет имеет множество ошибок и неточностей, что делает его использование невозможным.

Третьей проблемой описанной ранее работы являлось неправильное использование технологии Oversampling. Как выяснилось в ходе исследования разработчики модели использовали библиотеку не только на обучающей выборке, но и на тестовой. Это означает что оценка полученных результатов является искажённой и не соответствует действительности.

Таким образом для решения проблемы был собран новый датасет из 10 магистерских диссертаций. Кроме этого, из него были удалены наиболее зависимые от оформления признаки, что решило две проблемы: большую размерность датасета и зависимость от правильности оформления. Итоговая структура датасета представлена на рисунке 38.

```

RangeIndex: 906 entries, 0 to 905
Data columns (total 10 columns):
#   Column                Non-Null Count  Dtype
---  -
0   text                   906 non-null    object
1   count_of_sp_sb1        906 non-null    int64
2   count_sb1              906 non-null    int64
3   uppercase              906 non-null    int64
4   lowercase              906 non-null    int64
5   last_sb1               664 non-null    object
6   first_key              552 non-null    object
7   PrevElementMark        900 non-null    object
8   CurElementMark         906 non-null    object
9   NextElementMark        900 non-null    object
dtypes: int64(4), object(6)
memory usage: 70.9+ KB

```

Рисунок 38 – Структура нового, собранного датасета

После исследования исходной выборки и ГОСТов был обновлён перечень классов текстовых структурных элементов. Из них были удалены такие классы как рисунок, таблица и формула, т. к. они определяются на этапе извлечения данных и не являются текстовыми. Обновлённый перечень классов представлен в таблице 10.

Таблица 10 – Обновлённые классы текстовых структурных элементов

| Наименование класса | Описание |
|---------------------|-------------------------------|
| b1 | Заголовок 1 уровня |
| b2 | Заголовок 2-3 уровней |
| c1 | Обычный абзац |
| c2 | Абзац перед списком |
| d2 | Элемент перечисления (списка) |
| f1 | Подпись к таблице |
| f3 | Продолжение таблицы |
| h1 | Подпись к рисунку |
| j1 | Подпись к формуле |

Для получения лучшего эффекта глубина дерева была увеличена до 5, а их количество до 200.

В результате после обучения алгоритма с обновлённым датасетом и конфигурацией модели для визуального понимания была построена матрица ошибок Confusion Matrix. Она представлена на рисунке 39.

```
[[ 16  0  0  0  0  0  1  0]
 [  0  9  0  0  0  0  0  0]
 [  0  0 101  0  0  0  5  0]
 [  0  0  0 13  0  0  0  0]
 [  0  0  0  2 127  0  0  0]
 [  0  0  0  0  0  3  0  0]
 [  0  0  0  0  0  0  1  0]
 [  0  0  0  0  0  0  0 40]]
```

Рисунок 39 – Матрица ошибок оптимизированного классификатора

На рисунке 40 приведен отчёт по классификации, построенный при помощи библиотеки Sklearn и метода `classification_report`.

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 0.94 | 0.97 | 17 |
| 1 | 1.00 | 1.00 | 1.00 | 9 |
| 2 | 1.00 | 0.95 | 0.98 | 106 |
| 3 | 0.87 | 1.00 | 0.93 | 13 |
| 4 | 1.00 | 0.98 | 0.99 | 129 |
| 5 | 1.00 | 1.00 | 1.00 | 3 |
| 6 | 0.14 | 1.00 | 0.25 | 1 |
| 7 | 1.00 | 1.00 | 1.00 | 40 |
| accuracy | | | 0.97 | 318 |
| macro avg | 0.88 | 0.98 | 0.89 | 318 |
| weighted avg | 0.99 | 0.97 | 0.98 | 318 |

Рисунок 40 – Результаты тестирования классификатора

Из матрицы ошибок и полученным метрикам после тестирования видно, что классификатор практически не ошибается. Точность классификации составила 97%. За счёт высоких и практически одинаковых показателей метрик показатель F1 Score также высок.

Ниже приведена сводная таблица для более удобного сравнения первоначального и усовершенствованного алгоритма классификации. Для показателей Precision, Recall и F1-Score было взято среднее арифметическое по всем классам. Это представлено в таблице 11.

Таблица 11 – Сравнительная таблица метрик

| Метрика | Показатели исходного классификатора | Показатели разработанного классификатора |
|----------------------------|---|--|
| Accuracy | 96% | 97% |
| Macro Average Precision | 94% | 88% |
| Macro Average Recall | 91% | 98% |
| Macro Average F1-Score | 92% | 89% |
| Weighted Average Precision | 96% | 99% |
| Weighted Average Recall | 96% | 97% |
| Weighted Average F1-Score | 96% | 98% |
| Скорость обучения | 75 с | 87 с |
| Скорость предсказания | 0.00535 с | 0.00490 с |

Из данных таблицы следует что показатели значительно увеличились, с учётом большого изменение признаков. Показатель Macro Average Recall увеличился больше всех с 91% до 98%, что указывает на то, что выросла доля верно предсказанных объектов, которые модель определила к данному классу. Самым весомым событием стало увеличение общей точности классификатора с 96% до 97% с учётом искаженности первого показателя. Следует учитывать уменьшение скорости обучения с 75 до 87 с. за счёт усложнения модели (увеличения глубины дерева) и ускорение скорости предсказания с 0.00535 с. до 0.00490 с что имеет весомое значение при увеличении количества клиентов и соответственно нагрузки.

3.3 Api Gateway.

Чтобы изолировать друг от друга классификатор и парсер данных было решено создать многосервисную архитектуру. Для полноценной реализации которой необходимо было связать каждый сервис с другими и реализовать возможность балансировки и масштабирования каждого из сервисов.

Для создания взаимодействий было решено реализовать единую точку входа в сервисы, так называемый API GateWay.

Для решения поставленных задач было решено использовать следующие технологии:

- FastApi – для создания API каждого из сервисов и единой точки входа API GateWay,
- RabbitMQ – в качестве брокера сообщений,
- RPC server – для взаимодействия с RabbitMQ и реализации каждого запроса из очереди,
- Python 3.10 – как язык программирования,
- MySQL – в качестве базы данных.

Общая схема взаимодействия представлена на рисунке 41.

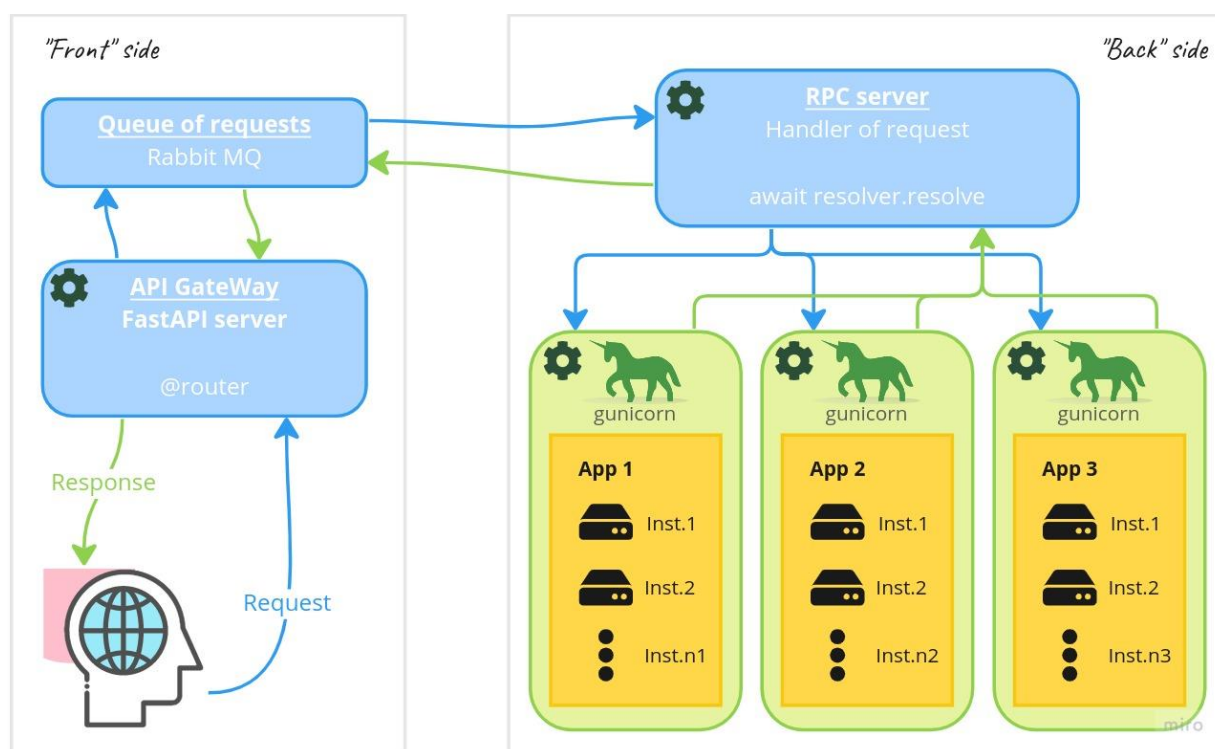


Рисунок 41 – Схема взаимодействия модулей с API GateWay

3.3.1 RabbitMQ

RabbitMQ – это брокер сообщений с открытым исходным кодом. Он маршрутизирует сообщения по всем базовым принципам протокола AMQP описанным в спецификации. Отправитель передает сообщение брокеру, а тот доставляет его получателю. RabbitMQ реализует и дополняет протокол AMQP.

Основная идея модели обмена сообщениями в RabbitMQ заключается в том, что producer (издатель) не отправляет сообщения непосредственно в очередь. На самом деле и довольно часто издатель даже не знает, будет ли сообщение вообще доставлено в какую-либо очередь.

Вместо этого издатель может отправлять сообщения только на обмен. С одной стороны, обмен получает сообщения от издателей, а с другой – отправляет их в очереди. Обмен должен точно знать, что делать с полученным сообщением.

Схема работы RabbitMQ представлена на рисунке 42

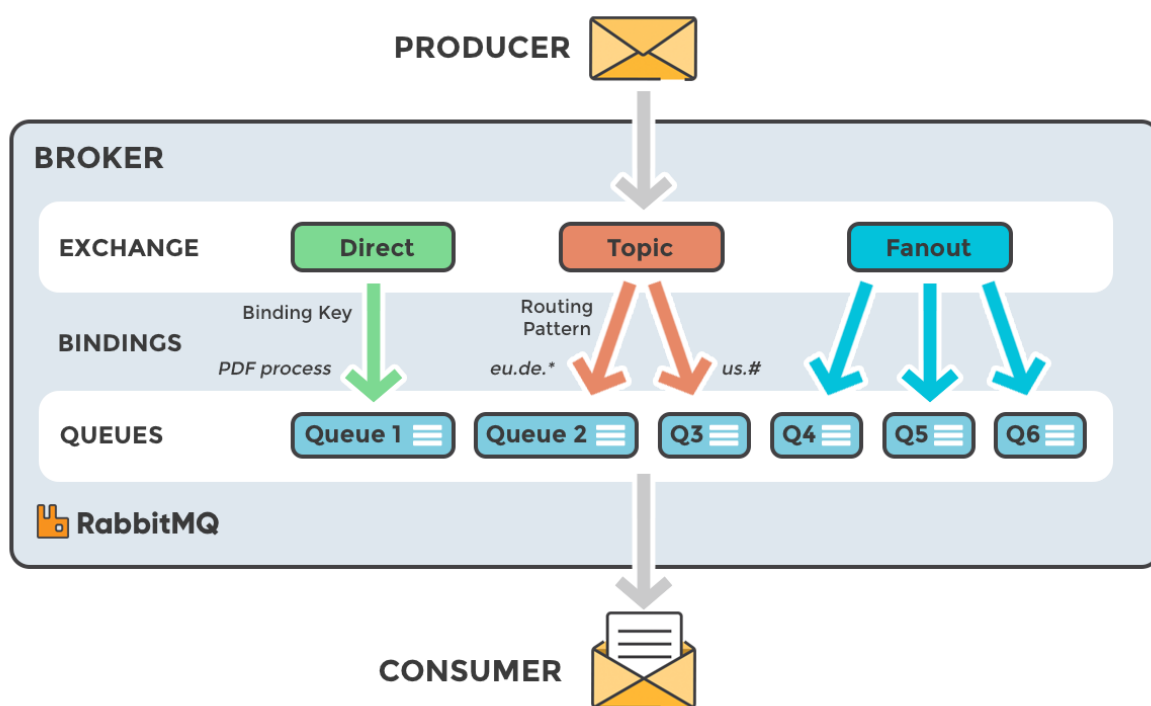


Рисунок 42 – Схема работы RabbitMQ

Кратко работу RabbitMQ можно описать следующим образом:

- издатель отправляет сообщение определенному обменнику,

- обменник, получив сообщение, маршрутизирует его в одну или несколько очередей в соответствии с правилами привязки между ним и очередью,
- очередь хранит ссылку на это сообщение. само сообщение хранится в оперативной памяти или на диске,
- как только потребитель готов получить сообщение из очереди, сервер создает копию сообщения по ссылке и отправляет,
- потребитель получает сообщение и отправляет брокеру подтверждение,
- брокер, получив подтверждение, удаляет копию сообщения из очереди. затем удаляет из оперативной памяти и с диска.

Как видно на рисунке 43 при запуске проекта создаётся канал, содержащий 2 очереди.

Channel: 127.0.0.1:60935 -> 127.0.0.1:5672 (1)

▼ Overview

Message rates last minute ?

Currently idle

Details

| | | | | | | | |
|------------|-----------------|-----------------------|------|-------------------------|---|-----------------------|---|
| Connection | 127.0.0.1:60935 | State | Idle | Messages unacknowledged | 0 | Pending Raft commands | 0 |
| Username | guest | Prefetch count | 0 | Messages unconfirmed | 0 | | |
| Mode ? | C | Global prefetch count | 0 | Messages uncommitted | 0 | | |
| | | | | Acks uncommitted | 0 | | |

▼ Consumers (2)

| Consumer tag | Queue | Ack required | Exclusive | Prefetch count | Active ? | Activity status | Arguments |
|--|--|--------------|-----------|----------------|----------|-----------------|-----------|
| ctag1.7244940695164286a50f8f172ec6ea0e | amq_0x7dd5eb782715c6011ffd5a2b24af8a27 | ○ | ● | 0 | ● | up | |
| ctag1.e001798ce47e451e9dec1806e3754f27 | resolve | ● | ○ | 0 | ● | up | |

Рисунок 43 – Создаваемый проектом канал-обменник RabbitMQ

3.3.2 GateWay

Для реализации API использовался фреймворк FastAPI совместно с библиотекой pydantic.

В настоящий момент проект состоит из 4 модулей – микросервисов:

- Parser,
- Clasifier,
- NLP,
- Checker.

Каждый из сервисов может зависеть от результатов работы другого. Поэтому была создана общая точка входа, которая будет выполнять всю цепочку запросов сразу – Normocontroler.

На рисунке 44 представлены все энд поинты gateway api, а также все схемы входных данных.

Normcontrol GateWay 0.1.0 OAS3
/openapi.json

| | | |
|--------------------|---|---|
| normcontrol | | ^ |
| POST | /normocontroler Normocontroler | ▼ |
| clasifier | | ^ |
| POST | /clasifier/clasify Classify Element | ▼ |
| POST | /clasifier/thematics Nlp Thematics | ▼ |
| POST | /clasifier/structure Nlp Structure | ▼ |
| parser | | ^ |
| POST | /parser/parse_document Parse Document | ▼ |
| GET | /parser/parse_paragraph Parse Paragraph | ▼ |
| POST | /parser/parse_images Parse Images | ▼ |
| POST | /parser/parse_table Parse Table | ▼ |
| POST | /parser/parse_list Parse List | ▼ |
| check | | ^ |
| POST | /check/check Check Elements | ▼ |

Рисунок 44 – Все энд поинты сервиса

Далее будет более детально описан каждый из сервисов.

Микросервис Parser.

Parser представляет собой сервис, который служит для парсинга структурных элементов и их свойств.

Он имеет несколько endpoint, каждый из которых извлекает определённые структурные элементы и их атрибуты и один который извлекает все структурные элементы, в том порядке в котором они расположены внутри документа, и их свойства.

На входе каждый endpoint ожидает данные о пользователе, тип документа и путь до документа.

В результате работы он возвращает словарь с данными о структурных элементах.

Микросервис Clasifier.

Classifier представляет собой сервис, который по полученным на этапе парсинга данным классифицирует элемент по его типу.

В большинстве случаев с docx и odt простые абзацы ничем не отличаются от заголовков, подписей к таблицам и рисункам как в метаданных так и в визуальном плане. Если говорить про pdf документы, то в них вообще все структурные элементы представляют собой набор символов. Поэтому данный сервис и необходим для последующей проверки документов.

Как уже говорилось ранее на вход endpoint поступают извлечённые данные и данные о пользователе.

После работы сервис возвращает обновлённые, посредством добавления к ним метки класса, данные о структурных элементах.

Микросервис NLP.

NLP представляет собой вспомогательный сервис, который может быть полезен многим пользователем, так как служит для анализа текста на соответствие определённым критериям, например, таким как тема или цель исследования. Также имеется возможность проверки соответствия определённых глав отчёта другим главам, например, введение и вывода.

На входе каждый endpoint получает текстовые данные структурных элементов и данные о пользователе.

В результате работы сервис возвращает степень соответствия.

Микросервис Checker.

Checker представляет собой сервис, который выполняет главную для пользователя работу – проверку отчётов на оформление согласно ГОСТу. Для поддержки нескольких ГОСТов была создана база данных, которая постоянно взаимодействует с данным сервисом.

Специально для опытных пользователей существует endpoint для добавления своего правила или целого нормативного акта, после чего его тоже можно будет использовать.

На входе endpoint принимает данные обо всех структурных элементах и их свойствах, на основе которых и идёт дальнейшая проверка.

В результате работы сервис возвращает пользователю сформированный отчёт об ошибках и в целом проверке, а также рекомендации по их устранению.

3.3.3 RPC Server

Схема работы грс сервера следующая:

- когда клиент запускается, он создает анонимную эксклюзивную очередь обратного вызова,
- для запроса RPC клиент отправляет сообщение с двумя свойствами: `answer_to`, для которого задана очередь обратного вызова, и `corellation_id`, для которого установлено уникальное значение для каждого запроса,
- запрос отправляется в очередь `resolve`,
- RPC-сервер ожидает запросов в этой очереди. Когда появляется запрос, он выполняет работу и отправляет сообщение с результатом обратно Клиенту, используя очередь из поля `reply_to`,
- клиент ожидает данных в очереди обратного вызова. Когда появляется сообщение, оно проверяет свойство `corellation_id`. Если он соответствует значению из запроса, он возвращает ответ приложению.

Схематично процесс работы RPC представлен на рисунке 45.

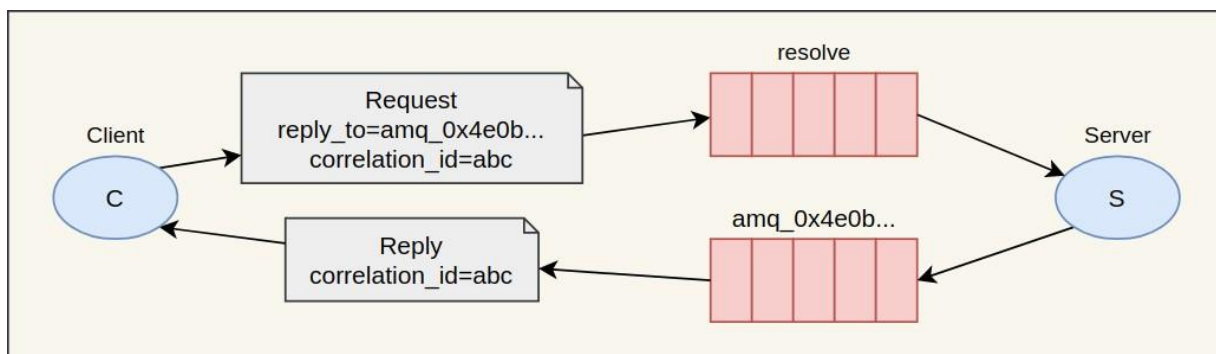


Рисунок 45 – Схема работы RPC сервера

В проекте грс сервер реализован при помощи библиотеки `aio_pika`. Листинг кода реализующий запуск сервера представлен на рисунке 46.

```

async def server() -> None:
    load_dotenv()
    rmq = os.environ.get("RMQ")
    __config = Config("settings.ini").to_dict()
    connection = await connect_robust(rmq)

    async with connection:
        channel = await connection.channel()
        rpc = await RPC.create(channel)
        await rpc.register(resolve.__name__, resolve, auto_delete=True)

        try:
            await asyncio.Future()
        finally:
            await connection.close()

if __name__ == "__main__":
    asyncio.run(server())

```

Рисунок 46 – Программный код метода server

Функция server создаёт подключение к RabbitMQ, создаёт канал и rpc. После чего регистрирует функцию resolve для обработки всех сообщений. Далее функция ожидает появления новых сообщений. Функция resolve представлена на рисунке 47.

```

async def resolve(
    method: str,
    path: str,
    params: dict = None,
    data: dict = None,
    headers: dict = None
) -> Tuple[dict, int]:
    _, service_name, api = path.split("/")
    service_host = SERVICE_MAP[service_name]
    url = f"http://{service_host}/{api}"
    response, status_code = await make_request(
        url, method, params, data, headers
    )
    return response, status_code

```

Рисунок 47 – Программный код метода resolve

Функция принимает на входе данные о запрашиваемом сервисе, методе и внутреннюю передаваемую информацию.

В ходе работы метод определяет куда необходимо перенаправить запрос, выделяя необходимый url адрес из пула адресов и соответственно отправляет новый запрос со старыми данными через метод `make_request` предоставленный на рисунке 48.

```

async def make_request(
    url: str,
    method: str,
    params: dict = None,
    data: dict = None,
    headers: dict = None
) -> Tuple[dict, int]:

    if not headers:
        headers = {}
    if not data:
        data = {}

    async with ClientSession() as session:
        request = getattr(session, method)
        async with await request(
            url, params=params, json=data, headers=headers
        ) as response:
            response_json = await response.json()
            return response_json, response.status

```

Рисунок 48 – Программный код метода make_request

В результате работы приходит ответ запрашиваемого сервиса и его статус.

3.3.4 Результаты работы сервиса

В качестве критериев оценки полученных результатов выступают количество затрачиваемого на проверку времени и степень наличия человеческого фактора при проверке документа на нормоконтроль.

Для получения затрачиваемого времени на проверку человеком был проведён опрос экспертов. В ходе опроса выяснилось, что скорость проверки сильно зависит от её тщательности. При быстрой скорости проверки, которая занимает примерно полминуты на 1 страницу может быть упущено большое количество ошибок. При более детальной проверке скорость уменьшается примерно до 1–2 минут на страницу. В среднем получается, что эксперт на средний текстовый отчёт по НИР объёмом в 30 страниц затрачивает примерно

25–35 минут. При этом при нормоконтроле остаётся и очень сильно влияет человеческий фактор.

Для получения времени проверки документа сервисом было замерено время от получения отчёта на входе end-point normocontroler и после получения от него ответа. В результате тестирования скорость работы сервиса получилась в среднем 1 минуту 15 секунд на текстовый отчёт объемом 30 страниц. При работе сервиса человеческий фактор практически полностью исчезает и остаётся только в загрузке отчёта и выборе ГОСТа по которому необходимо провести проверку.

Все описанные критерии показаны в таблице 12.

Таблица 12 – Сравнение результирующих критериев

| Отчёт на 30 страниц | Эксперт | Сервис |
|--------------------------------|---------|--------|
| Время детальной проверки (мин) | 25 - 30 | 1,25 |
| Время быстрой проверки (мин) | 15 | 1,25 |
| Человеческий фактор | + | - |

В результате работы всего сервиса на выходе получается текст в формате JSON, где ключ представляет собой наименование представляет собой атрибут. Как видно на рисунке 49 атрибут result представляет собой результат проверки. Он может содержать следующие данные:

- success – содержит информацию о том какие атрибуты прошли проверку,
- warning – содержит информацию о том какие рекомендации ГОСТов были нарушены,
- error – содержит все найденные ошибки.

```

"2": {
  "element_class": "Paragraph",
  "page_breake_before": 0,
  "keep_lines_together": null,
  "keep_with_next": null,
  "outline_level": null,
  "indent": 1.25,
  "line_spacing": 0.36,
  "alignment": null,
  "mrgrg": 0,
  "mrglf": 0,
  "mrgtop": 0,
  "mrgbtm": 0,
  "result": {
    "Success": {
      "text_size 12.0": "OK!",
      "indent 1.25": "OK!",
      "italics 0": "OK!",
      "bold 0": "OK!",
      "text_size 16.0": "OK!"
    },
    "Warning": {
      "font_name Arial": "Warning!"
    },
    "Error": {}
  },
  "font_name": [
    "Arial"
  ],
  "text_size": [
    14
  ],
  "text": "В ходе предыдущей исследовательской ...",
  "lowercase": false,
  "uppercase": false,
  "last_sbl": ".",
  "first_key": "",
  "bold": false,
  "italics": false,
  "underlining": 0,
  "sub_text": 0,
  "super_text": 0,
  "color_text": (255, 255, 255),
  "no_change_fontname": true,
  "no_change_text_size": true,
  "current_element_mark": "c1",
},

```

Рисунок 49 – JSON текст, представляющий собой результат работы всего сервиса

3.4 Вывод по главе 3

В результате работы были созданы модули для:

- классификации текстовых структурных элементов, на основе алгоритма CatBoost;
- унифицированные классы представляющие структурные элементы текстовых документов и их свойства;
- единая точка входа в сервис, которая обеспечивает взаимодействие между всеми модулями.

Оценка и тестирование классификатора на 30 текстовых отчётах показали очень хорошие результаты с показателем F1-score в 97%. Основные проблемы при классификации возникли в определении класса заголовков 1 и 2 уровней в связи с похожестью их содержания и оформления.

Разработанные унифицированные классы позволяют хранить и передавать информацию о структурных элементах и их свойствах независимо от формата исходного файла, откуда были получены первичные данные. Это позволит в будущем добавлять поддержку новых форматов файлов без изменения других модулей.

ЗАКЛЮЧЕНИЕ

В ходе исследования были проанализированы существующие решения в сфере нормоконтроля документации и описаны их проблемы, изучены форматы текстовых документов, их различия и особенности, выделены основные проблемы, возникающие при парсинге PDF документов.

В результате работы были разработаны алгоритмы для составления строк из символов pdf документа и алгоритм для выделения из строк параграфов. Вместе с выделением параграфов также образуются их собственные свойства и атрибуты посредством наследования их сначала от символов, а затем и от составляющих их строк. В результате получилось извлечь следующие атрибуты текстовых параграфов:

- текст,
- количество символов, спецсимволов и слов,
- шрифт и кегль,
- курсив, жирность,
- межстрочный интервал,
- отступ от красной строки,
- однородность шрифта и кегля
- uppercase и lowercase,
- позицию абзаца на странице,
- класс первого и последнего символа абзаца.

Был создан классификатор структурных элементов с точностью метрики F1-Score в 98%. Для классификатора был собран новый датасет, содержащий в себе признаки текстовых структурных элементов

Для унификации структурных элементов был разработан набор классов, позволяющий адаптировать сервис к различным форматам.

По итогам исследования был автоматизирован процесс нормоконтроля отчётной документации с увеличением скорости проверки более чем в 15 раз, исключением человеческого фактора в ходе проверки и точностью результата

не меньше, чем при работе эксперта. На выходе работы сервиса образуется JSON строка, которую можно использовать для составления отчёта по проверки и дальнейшего исправления ошибок.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. ISO. Document management—Portable document format—Part 1:PDF 1.7. ISO 32000—1:2008, International Organization for Standardization, Geneva, Switzerland, 2008.
http://www.adobe.com/content/dam/Adobe/en/devnet/acrobat/pdfs/PDF32000_2008.pdf.
2. Moore, R. (2009). Ongoing efforts to generate “tagged PDF” using pdfTEX. Towards a Digital Mathematics Library. Grand Bend, Ontario, Canada, July 8-9th, 2009, 125-131.
3. Воронкова Полина Николаевна, Французова Юлия Вячеславовна Обеспечение качества конструкторской документации за счет средств автоматизации нормоконтроля // Известия ТулГУ. Технические науки. 2017. №8-1. URL: <https://cyberleninka.ru/article/n/obespechenie-kachestva-konstruktorskoj-dokumentatsii-za-schet-sredstv-avtomatizatsii-normokontrolya> (дата обращения: 10.05.2023).
4. Соколов Александр Александрович, Дворянкин Александр Михайлович, Ужва Алексей Юрьевич Разработка метода автоматизации процесса нормоконтроля технической документации // Известия волгоградского государственного технического университета. 2013 (дата обращения: 12.05.2023).
5. Стариченко Борис Евгеньевич, Устинов Максим Андреевич Программа автоматизации контроля оформления текстовых документов // Педагогическое образование в России. 2018. №8. URL: <https://cyberleninka.ru/article/n/programma-avtomatizatsii-kontrolya-oformleniya-tekstovyh-dokumentov> (дата обращения: 14.05.2023).
6. Самойлова, И. А. Некоторые особенности поэтапного алгоритма программы для проверки дипломных работ на нормоконтроль / И. А. Самойлова. — Текст : непосредственный // Молодой ученый. — 2016. — № 16 (120). — С. 51-53. — URL: <https://moluch.ru/archive/120/33282/> (дата обращения: 16.02.2023).

7. Приложение «Электронный нормоконтролер» [Электронный ресурс]. URL: <https://play.google.com/store/apps/details?id=com.vildanov.normativecontroler&hl=ru> (дата обращения: 20.02.2023).
8. Nail Nasyrov, Mikhail Komarov, Petr Tartynskikh, Nataliya Gorlushkina. Automated formatting verification technique of paperwork based on the gradient boosting on decision trees // *Procedia Computer Science*, Volume 178, 2020, pp. 365-374, ISSN 1877-0509 (дата обращения: 24.02.2023).
9. Марцинкевич В. И., Ларионова Г. С., Терещенко В. В., Ситникова К. А. Анализ возможностей парсинга электронных текстовых документов для автоматизации нормоконтроля // *Экономика. Право. Инновации*. - 2022. - Т. 1, №.3. - С. 39-49 (дата обращения: 25.02.2023).
10. Тартынских П.С., Комаров М.С., Насыров Н.Ф. Подходы к автоматизированному анализу оформления электронных документов формата docx//Сборник тезисов докладов конгресса молодых ученых. Электронное издание. – СПб: Университет ИТМО, 2020. Электронное издание. – [2020, электронный ресурс]. – URL: <https://kmu.itmo.ru/digests/article/4592>, своб.— 2020 (дата обращения: 28.02.2023).
11. Endignoux, G., Levillain, O., & Migeon, J. Y. (2016, May). Caradoc: A pragmatic approach to pdf parsing and validation. In 2016 IEEE Security and Privacy Workshops (SPW) (pp. 126-139). Ieee (дата обращения: 01.03.2023).
12. Parinov, S. (2017). Extraction and visualisation of citation relationships and its attributes for papers in PDF. *International Journal of Metadata, Semantics and Ontologies*, 12(4), 195-203 (дата обращения: 03.03.2023)..
13. Xu, C., Tang, Z., Tao, X., Li, Y., & Shi, C. (2013, March). Graph-based layout analysis for pdf documents. In *Imaging and printing in a web 2.0 world iv* (Vol. 8664, pp. 34-41). SPIE (дата обращения: 05.03.2023).
14. Gao, L., Tang, Z., Lin, X., Liu, Y., Qiu, R., & Wang, Y. (2011, June). Structure extraction from PDF-based book documents. In *Proceedings of the 11th*

annual international ACM/IEEE joint conference on Digital libraries (pp. 11-20) (дата обращения: 07.03.2023)..

15. Hao, L., Gao, L., Yi, X., & Tang, Z. (2016, April). A table detection method for pdf documents based on convolutional neural networks. In 2016 12th IAPR Workshop on Document Analysis Systems (DAS) (pp. 287-292). IEEE (дата обращения: 09.03.2023).

16. Shao, M., & Futrelle, R. P. (2006). Recognition and classification of figures in PDF documents. In Graphics Recognition. Ten Years Review and Future Perspectives: 6th International Workshop, GREC 2005, Hong Kong, China, August 25-26, 2005, Revised Selected Papers 6 (pp. 231-242). Springer Berlin Heidelberg (дата обращения: 12.03.2023)..

17. Kahu, S. Y. (2020). Figure extraction from scanned electronic theses and dissertations (Doctoral dissertation, Virginia Tech) (дата обращения: 20.03.2023).

18. Schäfer, U., & Kiefer, B. (2011). Advances in deep parsing of scholarly paper content. In Advanced Language Technologies for Digital Libraries: International Workshops on NLP4DL 2009, Viareggio, Italy, June 15, 2009 and AT4DL 2009, Trento, Italy, September 8, 2009 (pp. 135-153). Springer Berlin Heidelberg (дата обращения: 25.03.2023).

19. Chao, H., & Fan, J. (2004). Layout and content extraction for pdf documents. In Document Analysis Systems VI: 6th International Workshop, DAS 2004, Florence, Italy, September 8-10, 2004. Proceedings 6 (pp. 213-224). Springer Berlin Heidelberg (дата обращения: 27.03.2023).

20. Hsiao, W. F., Chang, T. M., & Thomas, E. (2014). Extracting bibliographical data for PDF documents with HMM and external resources. Program, 48(3), 293-313 (дата обращения: 30.03.2023).

21. Wang, X., & Liu, J. C. (2017, November). A font setting based Bayesian model to extract mathematical expression in PDF files. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) (Vol. 1, pp. 759-764). IEEE (дата обращения: 02.04.2023).

22. Zhang, X., Gao, L., Yuan, K., Liu, R., Jiang, Z., & Tang, Z. (2017, November). A symbol dominance based formulae recognition approach for pdf documents. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR) (Vol. 1, pp. 1144-1149). IEEE (дата обращения: 04.04.2023).
23. Ramakrishnan, C., Patnia, A., Hovy, E., & Burns, G. A. (2012). Layout-aware text extraction from full-text PDF of scientific articles. Source code for biology and medicine, 7, 1-10 (дата обращения: 06.04.2023).
24. Pdfplumber documentation [Электронный ресурс]. URL: <https://github.com/jsvine/pdfplumber> (дата обращения: 07.04.2023).
25. Catboost Documentation [Электронный ресурс]. – URL: <https://catboost.ai/docs/concepts/parameter-tuning.html> (дата обращения: 10.04.2023).
26. Документация библиотеки Sklearn [Электронный ресурс]. – URL: <https://scikit-learn.org/stable/> (дата обращения: 12.04.2023).
27. Документация библиотеки Imblearn [Электронный ресурс]. – URL: <https://imbalanced-learn.org/stable/> (дата обращения: 20.04.2023).
28. Machine Learning —Multiclass Classification with Imbalanced Dataset [Электронный ресурс]. URL: <https://towardsdatascience.com/machine-learning-multiclass-classification-with-imbalanced-data-set-29f6a177c1a> (дата обращения: 27.04.2023).
29. Confusion Matrix for Your Multi-Class Machine Learning Model [Электронный ресурс]. URL: <https://towardsdatascience.com/confusion-matrix-for-your-multi-class-machine-learning-model-ff9aa3bf7826> (дата обращения: 07.05.2023).
30. Документация FastAPI [Электронный ресурс]. URL: <https://fastapi.tiangolo.com/> (дата обращения: 09.05.2023)
31. Документация RabbitMQ [Электронный ресурс]. URL: <https://www.rabbitmq.com/> (дата обращения: 15.05.2023)

32. Документация pydantic [Электронный ресурс]. URL: <https://pypi.org/project/pydantic/> (дата обращения: 16.05.2023)

33. Remote procedure call (RPC) [Электронный ресурс]. URL: <https://www.rabbitmq.com/tutorials/tutorial-six-dotnet.html> (дата обращения: 16.05.2023)