



Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра Системного Программирования

Богатенкова Анастасия Олеговна

Извлечение иерархической логической структуры из текстовых документов в формате docx

Выпускная квалификационная работа

Научный руководитель:
Гомзин Андрей Геннадьевич
Научный консультант:
Козлов Илья Сергеевич

Москва, 2021

Аннотация

Извлечение иерархической логической структуры из текстовых документов в формате docx

Богатенкова Анастасия Олеговна

Многие документы имеют логическую структуру, выделение которой может помочь при решении задач автоматизированного анализа документов. В настоящее время большое количество документов создаётся и хранится в формате docx. Однако данный формат позволяет описывать в основном только физическую структуру документа, то есть описывается то, как выглядит документ. В данной работе представлен один из способов, которым можно выделять из подобных документов логическую структуру в виде дерева на примере технических заданий.

Abstract

Hierarchical logical structure extraction from text documents in docx format

Bogatenkova Anastasiia Olegovna

Many documents have a logical structure. Logical structure extraction can solve some problems of automated document analysis. Currently, a large number of documents are created and stored in docx format. However, this format describes mostly the physical document structure, that is its appearance. This work presents one of the ways of logical tree structure extraction from such documents using the example of technical specifications.

Содержание

1	Введение	5
2	Постановка задачи	8
3	Обзор существующих решений	9
3.1	Извлечение иерархической логической структуры с ограничением на глубину дерева	9
3.2	Извлечение логической структуры из документов в формате docx	11
3.3	Извлечение неограниченной логической структуры на основе правил . .	12
3.4	Выводы	13
4	Исследование и построение решения задачи	14
4.1	Формализация логической структуры ТЗ	14
4.2	Разработка метода построения иерархической логической структуры из ТЗ	16
4.3	Разработка метода определения типа параграфов документа	19
4.3.1	Извлечение признаков	19
4.3.2	Обучение классификатора	21
4.3.3	Постобработка	21
4.4	Разработка метода сравнения параграфов документа	22
4.4.1	Извлечение признаков	23
4.4.2	Обучение классификатора	24
4.5	Создание обучающего набора документов	26
4.5.1	Создание изображений для разметки из docx файла	27
4.5.2	Манифесты для классификаторов	30
4.5.3	Набор данных для разметки и процесс разметки	30
4.6	Оценка качества разработанного метода извлечения иерархической структуры	32
4.7	Реализация метода извлечения структуры	32
4.7.1	Классификатор параграфов (tz_classifier)	34
4.7.2	Классификатор пар параграфов (pair_classifier)	34
4.7.3	Конструктор дерева документа (tree_constructor)	35

4.8	Сравнение с существующим методом	37
4.9	Выводы	38
5	Описание практической части	40
5.1	Существующие библиотеки по работе с форматом docx	40
5.1.1	python-docx	40
5.1.2	docx2python	41
5.1.3	pydocx	42
5.1.4	Выводы	42
5.2	Составляющие docx-файла	42
5.3	Некоторые понятия, используемые в документации формата	44
5.3.1	Тело документа (body)	44
5.3.2	Параграф (paragraph)	44
5.3.3	Текстовый элемент (run)	45
5.3.4	Стиль (style)	46
5.3.5	Абстрактная и непосредственная нумерация (abstract numbering and numbering)	48
5.4	Реализация обработчика docx документов	51
5.5	Тестирование обработчика docx документов	54
5.6	Выводы	55
6	Заключение	56
	Список литературы	58
	Приложение 1	61
	Приложение 2	67

1 Введение

В мире каждый год создаётся большое количество документов, таких как законы, судебные решения, технические задания, договоры и т. д. Объём документов многократно превышает возможности их неавтоматического анализа, например, по состоянию на 21.03.2021 список опубликованных правовых актов Правительства Российской Федерации составляет 32369¹. Таким образом, актуальной становится задача автоматической обработки и анализа документов, в том числе с использованием методов анализа естественного языка [1, 2, 3].

Однако, прежде чем перейти к анализу документов, необходимо представить документ в виде, удобном для автоматической обработки. Для этих целей в ИСП РАН разрабатывается программный модуль `docreader`, основанный на проекте `dedoc`², также разрабатываемом в ИСП РАН. Оба этих проекта написаны на языке `python`, так как в процессе анализа документов активно используется машинное обучение, а язык `python` хорошо подходит для этих целей.

Первым шагом в процессе анализа документа является выделение текста и структуры документа. Для документов можно определить три основных типа структуры, которая из них выделяется: физическая, логическая и семантическая [4].

1. Физическая структура документа описывает то, как выглядит документ. Другими словами, физическая структура оперирует терминами «символ», «набор символов», «строка», «блок текста», «страница». Помимо текстового содержимого, объектами физической структуры могут быть изображения, графики, таблицы и т. д. Кроме того, физическая структура описывает свойства символов (например, жирность или размер шрифта), строк (отступ от краев страницы) и текстовых блоков (расстояние между строками), описывается положение элементов на странице. Физическая структура не подразумевает выделения функций отдельных частей документа, их порядка чтения или смысла.
2. Логическая структура документа описывает разбиение документа на компоненты. Компоненты документов могут, в свою очередь, состоять из меньших компонент. Таким образом логическая структура документа может быть представлена в виде

¹<http://publication.pravo.gov.ru/SignatoryAuthority/government>

²<https://github.com/ispras/dedoc>

дерева[5, р. 101-116]. Разные виды документов могут разбиваться на разные типы компонентов. Например, научные статьи разбиваются на секции, законы состоят из глав, главы разбиваются на статьи и т. д.³

Логическая структура не анализирует смысл предложений, их взаимодействие в семантическом смысле (например, одно предложение ссылается на другое). Логическая структура строится на основе известной физической структуры и правил, задаваемых конкретным доменом.

3. Семантическая структура связана с задачей понимания содержимого текста. Например, можно определить взаимодействие именованных сущностей, упомянутых в тексте. В учебном пособии [6, р. 113] в качестве логических компонентов документа выделяются абзацы, а в качестве семантических составляющих сверхфразовые единства. Задачами выявления семантики текста занимается отдельная область – обработка естественного языка.

Как правило, разные виды структуры связаны. Например, начало новой главы часто выделяется жирностью и размером текста, а фразы, связанные семантически, обычно помещаются в один логический компонент. В то же время эта связь не является жёсткой, потому что одной и той же логической структуре может соответствовать различная физическая [4]. Для решения таких не полностью формализованных задач удобно использовать машинное обучение.

Документы могут быть представлены различными форматами, такими как pdf, djvu, odt. Также большое количество документов создается, редактируется и хранится в формате docx. В связи с этим встаёт вопрос о возможности автоматической обработки docx документов.

Для обработки docx документов можно использовать библиотеку для языка C#⁴, однако она позволяет работать с документами на низком уровне устройства формата docx. Проекты ИСП РАН, разрабатываемые для анализа документов, написаны на языке программирования python, поэтому использование библиотеки C# как части проекта не является удобным для реализации. Язык python также позволяет работать с docx

³Структура законов регулируется "Методическими рекомендациями по юридико-техническому оформлению законопроектов"

⁴<https://github.com/OfficeDev/Open-XML-SDK>

документами на низком уровне анализа xml составляющих документа⁵.

Существующие библиотеки по работе с docx документами на языке python (см. секцию 5.1) предназначены в первую очередь для создания и редактирования документов, а не для автоматического чтения. При чтении документов библиотеки могут игнорировать стили документа и символы нумерации в списках, однако эта информация крайне важна с точки зрения извлечения метаданных, необходимых для автоматического извлечения структуры. Поэтому актуализируется задача написания обработчика docx документов на языке программирования python.

Docx формат позволяет логически структурировать документы с помощью иерархии стилей заголовков. Однако отражение логической структуры не является основной целью данного формата, который предназначен для описания физической структуры документа – его визуального представления. Офисные приложения предоставляют возможность создания структурированного документа без использования специальных возможностей. Например, можно вручную прописывать элементы списков или содержание документа. Поэтому логическая структура в docx документах далеко не всегда присутствует явным образом. Таким образом, автоматическое извлечение логической структуры из docx документов конкретного домена (технические задания, законы, научные статьи) становится актуальной задачей.

Распространённым типом документов являются документы с требованиями к проектам в машиностроении, производстве и бизнесе. Такие документы относятся к техническим заданиям и могут не соответствовать требованиям, предъявляемым государственными стандартами. Тем не менее, технические задания имеют определённую логическую структуру, существуют шаблоны технических заданий в формате docx⁶. Точного определения структуры технического задания не было найдено, так как её исследование не является широко распространённым явлением. Определение возможной структуры и автоматическое извлечение этой структуры из технических заданий в формате docx может быть использовано для упрощения работы с техническими заданиями.

⁵<https://beautiful-soup-4.readthedocs.io/en/latest/>

⁶http://technicaldocs.ru/rocr19/шаблоны/техническое_задание

2 Постановка задачи

Техническое задание – документ, содержащий требования, предъявляемые к составным частям какого-либо проекта. Технические задания могут создаваться для разных целей и иметь разную структуру. Структура технического задания слабо исследована, в работах по анализу ТЗ [7, 8] предложено иерархическое устройство технических заданий. Ограничимся рассмотрением технических заданий на автоматизированные системы (АС). Технические задания на АС – слабо структурированный документ, описывающий требования к АС и регламентирующий процесс ее создания.

Задача состоит в формализации понятия иерархической логической структуры для технического задания на АС и автоматизации получения такой структуры из текстов технических заданий. Необходимо разработать и реализовать метод извлечения логической структуры из технических заданий, представленных в формате docx.

В рамках поставленной задачи необходимо выполнить следующее:

- Формально описать извлекаемую иерархическую логическую структуру;
- Реализовать метод построения иерархической логической структуры из технических заданий в формате docx;
- Провести оценку качества реализованного метода;
- Реализовать извлечение текста и необходимых метаданных из документов в формате docx. Встроить реализованный метод в открытый проект по обработке документов⁷.

⁷<https://github.com/ispras/dedoc>

3 Обзор существующих решений

Логическую структуру документа можно извлекать различными способами. Помимо отличий в определении логической структуры (которая не обязательно может быть иерархической), методы могут различаться доменами, для которых они разрабатываются, а также типами документов, которые обрабатываются. Так, наиболее распространенный домен для исследований – это научные статьи, при этом большая часть методов работает с документами в pdf формате, так как этот формат широко распространен и практически во всех системах отображается одинаковым образом. Однако документы в формате PDF не всегда имеют текстовый слой, а если имеют, то он может быть некорректен. Кроме того, из таких документов сложно вычленивать информацию о визуальном представлении строк, например, информацию о шрифтах или выравнивании. Ограничение же на домен сильно сужает область исследований, так как статьи, как правило, имеют очень похожую логическую структуру, которая может существенно отличаться от документов других типов.

3.1 Извлечение иерархической логической структуры с ограничением на глубину дерева

В статье [3] решается задача извлечения логической структуры из pdf-документов. Логическая структура представляет собой последовательность вложенных друг в друга секций с текстовым содержимым. Вложенность ограничивается тремя уровнями, то есть секции делятся на секции верхнего уровня, подсекции и подподсекции. Авторы применяют свой метод на документах из двух доменов: научные статьи и документы запроса предложения. Помимо выделения иерархической логической структуры решалось еще несколько задач. Каждой секции назначалась семантическая метка, определялась тема секции и проводилась её суммаризация.

Логическая структура определялась в два этапа:

1. На первом этапе текстовые строки классифицировались на два класса: заголовок или текстовая строка. На данном этапе были испробованы разные методы: Метод Опорных векторов, Наивный Байес, Дерево решений, Рекуррентная нейронная сеть (показала лучший результат).

2. На втором этапе для строк, отклассифицированных как заголовки, определялся их уровень. Для классификации заголовков на три типа (секции верхнего уровня, подсекции и подподсекции) использовались нейронные сети – рекуррентная и сверточная нейронные сети. Сверточная нейронная сеть показала лучший результат.

Для классификаторов использовались визуальные и текстовые признаки, выделенные вручную (всего 16), а также признаки, основанные на символьных n-граммах. Классификаторы по отдельности тренировались на этих двух группах признаков, а затем проведено обучение на объединенном множестве признаков – в этом случае результаты получились лучше.

Научные статьи обрабатывались с помощью PDFLib TET⁸, признаки извлекались из выходного TETML файла. Секции трех уровней определялись с помощью содержания (автоматическая разметка). Всего в датасете было больше миллиона статей. Модели обучались на датасете из статей. Датасет бизнес-документов состоял из 250 документов, размеченных вручную. Модели были протестированы на размеченном датасете, результаты оказались несколько хуже результатов для статей, особенно в классификации секций.

Результаты данной статьи показывают, что при смене домена результат распознавания логической структуры может ухудшиться, так как научные статьи имеют свою специфику. Помимо этого анализ документов ограничивается анализом pdf-документов, результат обработки которых зависит от результата работы специализированной библиотеки. Формат docx предоставляет больше информации о визуальном представлении документов. И, наконец, структура документов ограничивается тремя уровнями вложенности, однако не все документы подходят под такое описание структуры.

В 2019 году проводились соревнования FinTOC [9], где из финансовых документов извлекалась структура в виде иерархии уровней заголовков документов. Максимальная глубина уровней равна пяти. Победители соревнования [10] извлекали необходимую структуру используя оглавление документов, а также систему правил, которые применялись для определения иерархии заголовков.

Сначала идентифицировались страницы, содержащие текст оглавления, затем в документе находились страницы, соответствующие заголовкам, указанным в оглавлении.

⁸<https://www.pdflib.com/products/tet/>

Последним шагом являлось выделение иерархии найденных заголовков, основанное на применении правил: анализировались такие признаки, как междустрочный интервал, отступ, шрифт, символы нумерации. Используемый подход позволил получить достаточно высокую точность, но низкую полноту, так как некоторые оглавления документов были неполными. Использование правил также ограничивает полноту реализованного метода.

3.2 Извлечение логической структуры из документов в формате docx

В статье [11] решается задача извлечения логической структуры из docx документов в виде классификации параграфов на 3 класса: секция, подсекция, содержимое (текст). При классификации использовались 4 алгоритма машинного обучения: Случайный лес, Метод Опорных векторов, Алгоритм k ближайших соседей и Наивный Байес. Для обучения классификаторов были вручную выделены 8 признаков: размер шрифта (нормализованный), отступ (есть или нет), точка в конце, несколько предложений, выравнивание (4 типа), жирность, нумерация, подчеркивание.

Набор данных был собран вручную из 50 документов (спецификации, отчеты, новости) и сбалансирован по числу элементов в каждом классе. Классы были размечены вручную. Обучение и тестирование классификаторов проводилось как на сбалансированном, так и на несбалансированном наборе данных. На сбалансированном наборе данных результаты в среднем оказались лучше, однако в некоторых случаях (например, для типа "содержимое") оказался лучше несбалансированный датасет.

В данной статье указывается то, что логическая структура docx-документов не всегда совпадает с их физической структурой, тем самым показывается актуальность задачи. Однако выделяется плоская структура из последовательности типизированных параграфов, которую можно использовать для построения более сложной и подробной структуры. Кроме того, возможности docx формата позволяют извлекать больше метаданных для параграфов.

3.3 Извлечение неограниченной логической структуры на основе правил

В патенте [12] предлагается структурированное представление электронного документа в виде дерева, элементами которого являются группы параграфов со схожими визуальными признаками (атрибутами). Если в документе присутствует информация о его визуальном представлении, система собирает эту информацию в специальной таблице тегов, которая хранит для каждого параграфа его тип.

Система выделяет структурные типы (заголовки, список, обычный параграф), которые удастся распознать, а также можно описать дополнительные типы для выделения. Структурные типы распознаются с помощью специальных правил и статистики, собранной системой. Каждому структурному типу присваивается уровень (с помощью методов сортировки). Для сортировки применяются специальные компараторы на основе правил (используется текст элемента, размер шрифта и т. д.). На основе выделенных элементов со структурными типами и уровнями строится дерево.

Обычные параграфы помечаются отдельно от остальных элементов документа. При построении дерева система читает последовательно параграфы исходного документа и если встречается обычный параграф, система добавляет этот параграф к уровню, на анализе которого она находится в данный момент. Содержимое параграфов помещается в листовые вершины. Если текущий параграф не является обычным параграфом и определённый уровень меньше текущего, осуществляется проход вверх по иерархии до тех пор, пока не уровень не сравняется с текущим, и происходит добавление нового узла, иначе (определённый уровень для данного узла больше или равен текущему уровню в дереве) добавляется новый узел без прохода по иерархии. Количество уровней структуры можно задать вручную для ограничения вложенности.

Предложенный метод позволяет выделить сложную иерархическую структуру в виде дерева, используется большое число визуальных и текстовых признаков. Однако данная система основана на правилах и поэтому не обладает гибкостью, которая необходима в некоторых случаях при обработке документов.

3.4 Выводы

Методы по извлечению логической структуры из документов различаются как по рассматриваемому домену и формату документа, так и по типу извлекаемой структуры. Данные методы предоставляют возможность классификации строк (или параграфов) документа и выстраивания некоторой структуры, однако следует выделить следующие особенности:

1. Рассмотренные методы ограничивают извлекаемую структуру: число уровней вложенности ограничивается заранее заданным числом. Описанный выше метод построения дерева произвольной глубины опирается на правила, прописанные вручную, что уменьшает универсальность работы метода. Поэтому построение дерева документа произвольной глубины без прописывания конкретных правил является некоторым улучшением существующих методов.
2. Документы, с которыми производится работа, представлены в формате pdf. Формат docx предоставляет больше возможностей по выделению метаданных из документов. Один из описанных выше методов работает с docx документами, однако использует лишь малую часть тех возможностей, которые предоставляет формат.
3. В главе 2 указано, что рассматриваются документы типа «техническое задание». Документы такого типа имеют свою специфическую структуру, которую необходимо исследовать и в соответствии с ней разработать метод обработки технических заданий.

4 Исследование и построение решения задачи

В соответствии с постановкой задачи необходимо исследовать структуру технических заданий на АС. После формализации логической структуры технического задания необходимо разработать и реализовать метод извлечения этой структуры. Поэтому решение задачи состоит из следующих пунктов:

1. Исследование структуры ТЗ и формализация логической структуры ТЗ;
2. Разработка метода построения формализованной логической структуры из ТЗ на АС в формате docx.

4.1 Формализация логической структуры ТЗ

В соответствии с поставленной задачей, необходимо формализовать логическую структуру, извлекаемую из технических заданий. Техническое задание является техническим документом, структура которого представима в виде дерева (см. главу 1). Иерархическое устройство технических заданий предложено в работах по анализу ТЗ [7, 8]. В результате изучения различных вариантов ТЗ и стандартов ГОСТ были выделены следующие составные части ТЗ:

1. Титульный лист, который находится в начале документа и содержит название, подписи и даты. Вместо титульного листа в документах может присутствовать только название на первой странице.
2. Содержание, которое может располагаться после титульного листа или отсутствовать.
3. Базовые структурные элементы (разделы, подразделы и т. д.), которые визуально выделяются и делят документ на крупные составные части. Структурные элементы составляют содержимое документа и вкладываются друг в друга.
4. Элементы списков, которые не отличаются визуально от основного текста, но структурируют текст с помощью нумерации.
5. Текстовые строки (или параграфы).

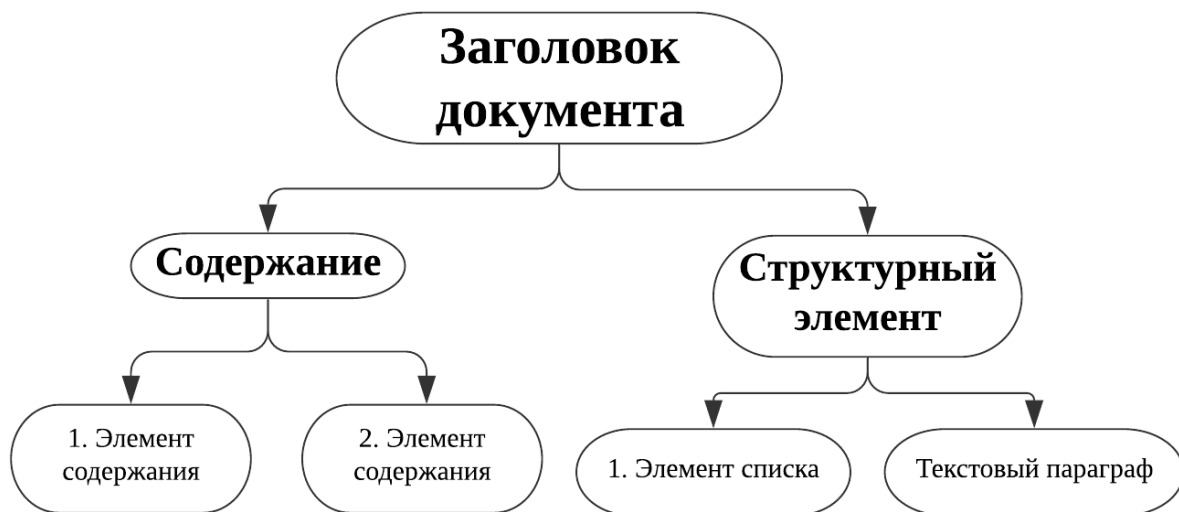


Рис. 1: Пример структуры технического задания

В результате изучения структуры ТЗ было решено представлять в виде дерева произвольной глубины, в котором узлы упорядочены, имеют один из пяти описанных выше типов и представляются следующим образом:

- Корень дерева содержит название документа, наиболее выделяющееся визуально. Остальные части названия являются дочерними узлами корня и располагаются в порядке чтения.
- Заголовок содержания (если содержание присутствует) является дочерним узлом названия-корня. Содержимое содержания является деревом с корнем в заголовке содержания. Иерархическая структура содержания соответствует структуре этого дерева.
- Структурные элементы являются дочерними узлами названия-корня (в порядке чтения). Если один из структурных элементов непосредственно вложен в другой (например, подраздел вложен в раздел), то он является дочерним узлом структурного элемента, в который он вкладывается.
- Элементы списков могут быть вложенными в структурные элементы, текстовые параграфы (если перед перечислением находится его описание с двоеточием в

конце) или другие элементы списков. В соответствии с этим определяется узел-предок элемента списка. Все элементы одного списка являются дочерними узлами одного предка.

- Простые текстовые параграфы могут вкладываться в структурные элементы или элементы списков. Таким определяется узел-предок для текстового параграфа.

На рисунке 1 изображён пример структуры технического задания, которое можно извлечь из документа.

Таким образом, необходимо автоматизировать выделение описанной выше древовидной структуры. Эта структура не является строго определённой, поэтому для её выделения логично использовать машинное обучение.

4.2 Разработка метода построения иерархической логической структуры из ТЗ

В секции 4.1 формализована логическая структура ТЗ, которую необходимо извлечь из документов в формате docx. Данная логическая структура представляет собой дерево произвольной глубины, в котором каждому узлу соответствует параграф документа с типом (заголовок, содержание, часть, элемент списка или текст).

Рассмотрим алгоритм построения дерева документа. Предположим, что в документе для каждой пары строк определена операция сравнения (больше, меньше или равно). Если первая строка больше второй, это означает, что первая строка «главнее» второй в некотором смысле.

На рисунке 2 показан пример документа, в котором для каждой строки указан её уровень. Например, строка с названием документа имеет уровень 0, а строка с текстом «Глава 1» имеет уровень 1. Это означает, что строка с названием документа «важнее» строки с названием главы, а значит больше по отношению сравнения строк документа. Далее сравнение строк документа будет описано подробнее, в текущем примере будем сравнивать строки в соответствии с их уровнем, указанным на рисунке 2. При этом, если уровень первой строки больше уровня второй строки, то вторая строка считается больше первой.

На основе сравнения пар строк структура дерева строится следующим образом:

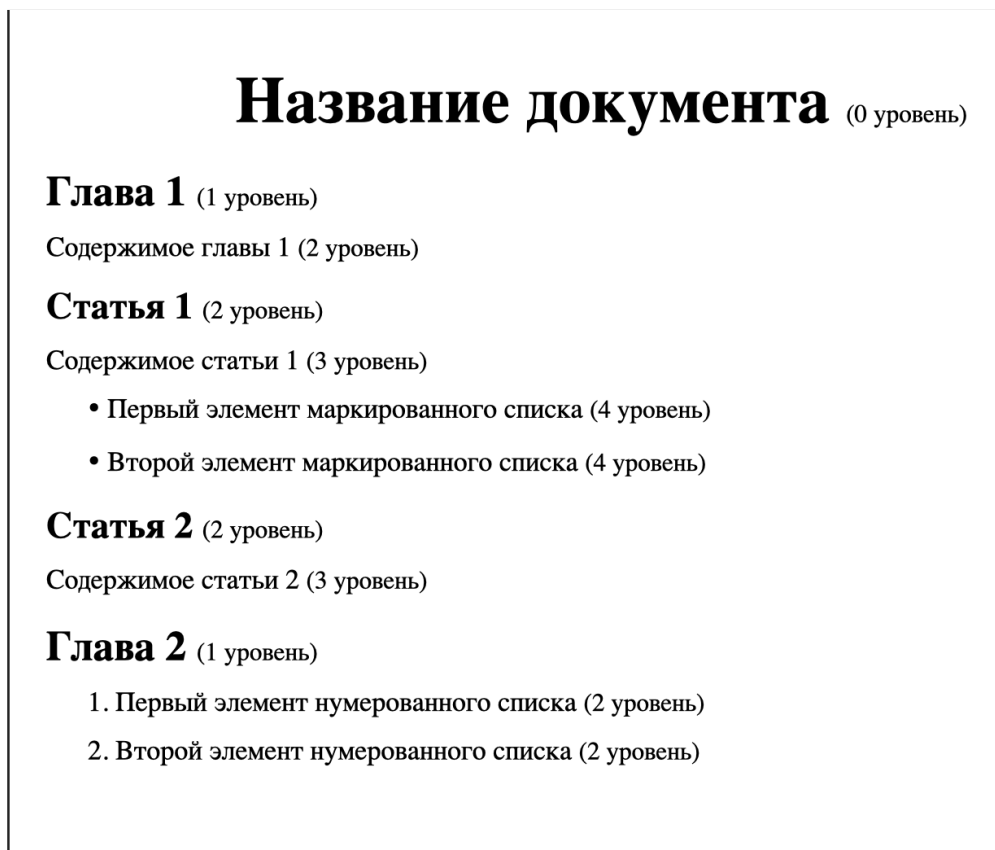


Рис. 2: Пример документа для извлечения структуры

1. Изначально дерево состоит из одного корня, который обозначает документ в целом и не имеет текстового содержимого. Корень используется для доступа к остальным элементам дерева.
2. Первая строка документа становится новым дочерним узлом корня.
3. Далее построение дерева происходит в процессе последовательного просмотра строк (или других составляющих) документа:
 - рассматривается следующая строка документа и сравнивается с предыдущей строкой;
 - если рассматриваемая строка равна предыдущей, в дерево добавляется новый узел, предком которого является предок узла предыдущей строки;
 - если рассматриваемая строка меньше предыдущей, в дерево добавляется новый узел, являющийся дочерним по отношению к узлу предыдущей строки;

- если рассматриваемая строка больше предыдущей, производится переход к узлу-предку предыдущей строки и сравнение текущей строки и строки узла-предка. Такие переходы и сравнения осуществляются до тех пор, пока текущая строка не окажется меньше или равна строке узла-предка. Тогда новый узел будет добавляться в соответствии с предыдущими двумя правилами относительно найденной строки. Если осуществился проход до корня дерева, к корню добавляется новый дочерний узел;
- после добавления узла рассматривается очередная строка и алгоритм повторяется сначала.

Следуя этому алгоритму, для документа, изображенного на рисунке 2, будет построено дерево, изображенное на рисунке 3.



Рис. 3: Пример построенного дерева на основе документа на рисунке 2

Используем данный алгоритм для построения логической структуры технических заданий в формате docx. При построении дерева для каждой вершины дерева необхо-

можно также определить её тип в соответствии с разработанной структурой: заголовок, содержание, часть, элемент списка или текст.

Формат docx позволяет описать физическую структуру документа, на основе которой необходимо построить логическую структуру. Документы в формате docx состоят из структурных единиц, называемых параграфами [13]. Поэтому основным элементом выделяемой логической структуры (вершиной дерева) логично сделать параграф. Прежде всего необходимо уметь программным образом «читать» документы в docx формате. Для реализации алгоритма построения дерева документа необходимо уметь сравнивать параграфы документа и определять их тип в соответствии с моделью ТЗ.

Таким образом, задача построения дерева требует решения следующих задач:

1. Разработка метода определения типа параграфов технического задания;
2. Разработка метода сравнения параграфов технического задания.

4.3 Разработка метода определения типа параграфов документа

Поставим задачу определения типа параграфа документа как задачу классификации параграфов на 5 типов: заголовок, содержание, часть, элемент списка или текст. Задачу классификации будем решать с помощью машинного обучения.

Общая схема метода определения типа параграфов документа выглядит следующим образом:

1. Извлечение метаданных для параграфов и формирование вектора признаков.
2. Обучение классификатора параграфов на основе извлеченных признаков.
3. Корректировка ответов классификатора с помощью постобработки.

Опишем каждый из шагов более подробно.

4.3.1 Извлечение признаков

С помощью реализованного обработчика docx документов (см. главу 5) для каждого параграфа можно выделить его текст и метаданные (размер шрифта, отступ от края страницы, выравнивание, жирность шрифта, курсив, подчеркивание, название стиля,

уровень вложенности для автоматических списков). На основе полученной информации составляются векторы признаков для параграфов.

Перечислим основные признаки, используемые для формирования векторов признаков:

- Признак оглавления (близость к параграфу с текстом «содержание» или «оглавление»).
- Признак ТЗ (близость к параграфу с текстом «техническое задание» или «приложение»).
- Признаки, полученные с помощью регулярных выражений для начала и конца строки (для обнаружения нумерации).
- Индикатор того, написан ли параграф только заглавными или строчными буквами.
- Признаки для списков: длина нумерации для параграфов-списков вида «1.1.1», индикатор может ли параграф быть продолжением списка, является ли параграф элементом автоматического списка, индикатор соответствия регулярным выражениям для списков.
- Номер параграфа, длина текста параграфа, число слов в тексте параграфа.
- Индикаторы того, если ли в параграфе жирный, курсивный, подчёркнутый текст.
- Выравнивание, отступ от левого края, размер шрифта текста.
- Признаки, полученные с помощью регулярных выражений для названий стилей («heading», «title», «list item», «contents»).
- Признаки трех предыдущих и трех следующих параграфов.

Таким образом, из списка обработанных документов (которые являются списками словарей с метаданными параграфов) получается матрица признаков для параграфов (с 234 признаками для каждого параграфа).

4.3.2 Обучение классификатора

В качестве классификатора использовался XGBoost [14], являющийся одним из самых эффективных алгоритмов машинного обучения на данный момент. Нейронные сети для решения поставленной задачи не подходят, так как признаки, используемые для обучения, очень разнородны по своим значениям.

В результате подбора параметров алгоритма и корректировки признаков после кросс-валидации были получены результаты, описанные в таблице 2 в строке «до постобработки». Разбиение документов на тренировочное и валидационное множества производилось 10 раз по группам, т. е. параграфы из одного документа не могли попасть в разные группы (и, соответственно, в разные множества).

Был проведён анализ ошибок классификатора, показанный в таблице 1. В таблице указаны 5 наиболее часто встречающихся пар классов, один из которых правильный, второй – предсказанный.

Правильная метка	Предсказанная метка	Суммарное число ошибок	Процент ошибок
item	part	45	27.78%
title	raw_text	29	17.90%
item	raw_text	23	14.20%
raw_text	title	15	9.26%
toc	raw_text	15	9.26%

Таблица 1: Ошибки классификатора типов параграфов

Классы item и part схожи между собой и их может перепутать даже человек. Однако класс title является очень важной составляющей документа, которая явно отличается от остальных частей, поэтому необходимо отличать его от raw_text. Таким образом, было решено добавить к классификатору постобработку для исправления некоторых ошибок.

4.3.3 Постобработка

Для исправления классификации заголовка (title) было сделано предположение, верное для документов типа ТЗ: заголовок встречается только в начале документа и преры-

вается, когда начинается параграф, отличающийся от обычного текста (`raw_text`).

Исходя из данного предположения предсказания классификатора меняются следующим образом: до начала тела документа (то есть до тех пор, пока не встретится `part`, `item` или `toc`) все параграфы помечаются как `title`. Остальные строки не могут быть `title`, поэтому их «вероятность» быть `title` зануляется.

Помимо этого, для пустых параграфов устанавливается класс «`raw_text`».

В результате сделанной постобработки значения достоверности и F1-меры на кросс-валидации выросли в соответствии с данными из таблицы 2, а процент неправильной замены `title` на `raw_text` снизился с 17.90% до 3.57%.

	Accuracy	F1-measure (macro)
До постобработки	0.95445	0.93003
После постобработки	0.96028	0.945

Таблица 2: Усреднённые значения достоверности и F1-меры на кросс-валидации

В процессе кросс-валидации было выявлено разбиение на тренировочное и валидационное множества, при котором достоверность предсказания была минимальной. Для такого разбиения была построена матрица ошибок (рисунок 4).

Исходя из матрицы ошибок можно заключить, что чаще всего путаются `item` и `part` как схожие структурные единицы. В силу сделанных предположений относительно заголовков, некоторые строки с простым текстом могут классифицироваться как заголовок. Однако строки заголовка, в которых содержится наиболее важная информация, классифицируются правильным образом.

4.4 Разработка метода сравнения параграфов документа

Поставим задачу сравнения параграфов документа как задачу классификации пар параграфов на 3 типа: больше, меньше или равно. Задачу классификации будем решать с помощью машинного обучения.

Общая схема метода построения иерархической структуры документа выглядит следующим образом:

1. Извлечение метаданных для пар параграфов и формирование вектора признаков.

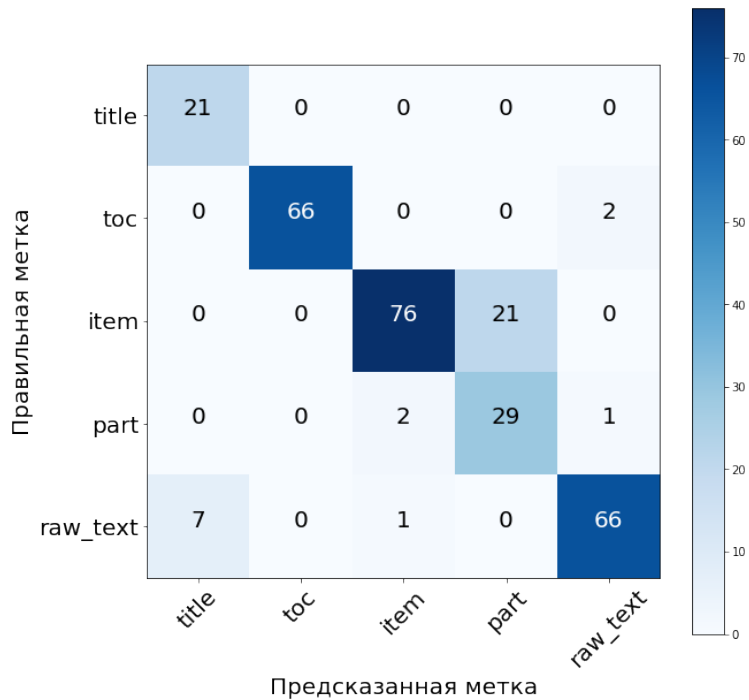


Рис. 4: Матрица ошибок классификатора типов параграфов

2. Обучение классификатора (компаратора) пар параграфов на основе извлеченных признаков.
3. Построение дерева документа с использованием обученного компаратора параграфов.

Опишем каждый из шагов более подробно.

4.4.1 Извлечение признаков

Для компаратора параграфов необходима информация о двух параграфах документа, то есть их текст и метаданные. Сравнивая метаданные, классификатор должен сделать вывод о том, какой из параграфов «больше», или параграфы «равны».

Для пары параграфов извлекались следующие признаки:

- Разница значений размеров шрифта и отступов от левого края.
- Разница значений индикаторов жирности, курсива, подчеркивания (а также разница процентных значений, если текст не выделен жирностью и т. д. целиком).

- Разница выравниваний и типов параграфов (каждому типу поставлено в соответствие число по приоритетам типов).
- Разница индикаторов соответствия параграфов стилям (вычислялись с помощью регулярных выражений).
- Разница индикаторов текста, написанного заглавными буквами.
- Разница индикаторов выравнивания по центру.
- Разница длин первых слов в параграфах.
- Индикатор начала списка после строки, заканчивающейся двоеточием.
- Разница индикаторов соответствия параграфов шаблонам конкретных типов списков (вычислялись с помощью регулярных выражений).
- Если оба параграфа – элементы списков типа 1.1.1, вычислялась разница между числом уровней нумерации.
- Разница индикаторов соответствия регулярным выражениям для конца строки (регулярные выражения упорядочены).
- Разница индикаторов соответствия регулярным выражениям для списков (регулярные выражения упорядочены, разница между индексами подошедших выражений).

Таким образом, из списка пар параграфов (списки из двух словарей с метаданными параграфов) получается матрица признаков для пар параграфов с 28 признаками для каждой пары.

4.4.2 Обучение классификатора

Аналогично случаю классификатора типов параграфов для классификации пар строк использовался XGBoost [14].

В результате подбора параметров алгоритма и корректировки признаков после кросс-валидации были получены результаты, описанные в таблице 3. Разбиение документов

Accuracy	F1-measure (macro)
0.9809	0.97848

Таблица 3: Усреднённые значения достоверности и F1-меры на кросс-валидации

на тренировочное и валидационное множества производилось 10 раз, в таблице представлены усреднённые значения достоверности и F1-меры.

Был проведён анализ ошибок классификатора, показанный в таблице 4. В таблице указаны пары классов, один из которых правильный, второй – предсказанный.

Правильная метка	Предсказанная метка	Суммарное число ошибок	Процент ошибок
equals	greater	59	27.83%
less	equals	39	18.40%
greater	equals	37	17.45%
less	greater	36	16.98%
equals	less	31	14.62%
greater	less	10	4.72%

Таблица 4: Ошибки классификатора пар параграфов

Результаты анализа ошибок логично интерпретируются: классификатор реже путает наиболее различающиеся классы «greater» и «less». Чаще всего классификатор предсказывает «greater» вместо «equals». Это объясняется тем, что часто визуальное незаметно выравнивание по центру, которое играет большую роль при классификации (из-за выравнивания по центру первого параграфа классификатор считает, что он больше второго). Ещё одной причиной неправильной классификации является сравнение текстовых строк и элементов списков – из-за недостатка разнообразия данных классификатор может считать, что текстовые строки важнее элементов списков. По этой же причине классификатор может путать «less» и «equals».

Может показаться странной относительно частая неправильная замена «less» на «greater». Это связано с тем, что не всегда элемент списка вида «1.» является более важным, чем элемент списка вида «1.1». Если эти элементы принадлежат разным спискам, то может оказаться наоборот. В силу этого и возникают ошибки классификации.

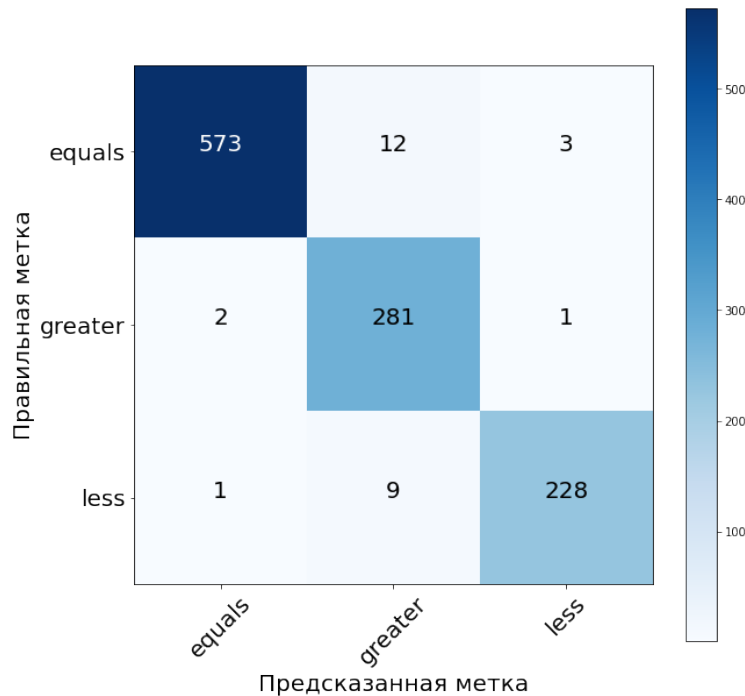


Рис. 5: Матрица ошибок классификатора пар параграфов

В процессе кросс-валидации было выявлено разбиение на тренировочное и валидационное множества, при котором F1-мера предсказания была минимальной. Для такого разбиения была построена матрица ошибок (рисунок 5). Результаты, указанные в матрице ошибок, не противоречат результатам из таблицы 4 и могут быть интерпретированы таким же образом. Следует отметить, что для построения дерева не нужна 100% точность работы компаратора пар строк. Наиболее критичной является неправильная классификация классов «greater» и «equals», так как при таких ответах классификатора не происходит сравнения с другими параграфами, а сразу создается новый узел в дереве.

4.5 Создание обучающего набора документов

В секции 4.2 задача извлечения структуры документа была разбита на две задачи классификации параграфов документа. Для решения этих задач необходим обучающий набор документов, на котором будут обучаться классификаторы.

Так как необходимо анализировать технические задания, написанные на русском

языке, нужно учитывать особенности создания документов такого типа. Язык написания, государственные стандарты и культура написания технических заданий оказывают сильное влияние на структуру технических заданий. Общедоступных наборов данных с подобными документами не было найдено, следовательно, необходимо его создать с помощью ручной разметки.

В процессе разметки удобно использовать изображения, которые можно классифицировать. При этом документ представляется как набор изображений, каждое изображение – это одна из страниц документа. Для того, чтобы разметить каждый параграф (то есть присвоить ему некоторую метку), можно создать набор изображений, в каждом из которых обведен в рамку один параграф документа. Тогда, присваивая метку изображению с параграфом, обведенным в рамку, мы присвоим метку этому параграфу.

Таким образом, процесс создания обучающего набора документов состоит из следующих шагов:

1. Создание изображений для разметки из docx файла. На каждом из изображений один из параграфов обведён в рамку. Изображение связано с параграфом с помощью уникального идентификатора параграфа.
2. Разметка каждого изображения аннотаторами с помощью специальной системы разметки⁹.
3. Получение меток для параграфов, соответствующих размеченным изображениям.

4.5.1 Создание изображений для разметки из docx файла

При создании в docx документах параграфов, обведенных в рамку, возникают некоторые особенности. При вставке рамки в текст документа, расстояние между параграфами увеличивается, вследствие чего параграфы визуально сдвигаются вниз. При этом, если обводить несколько параграфов одинаковым цветом, может произойти слияние рамок для двух параграфов в одну рамку.

Поэтому был разработан специальный алгоритм создания из docx документа набора изображений с параграфами, обведёнными в рамку (см. рисунок 6).

В общих чертах алгоритм состоит из следующих шагов:

⁹<https://github.com/dronperminov/ImageClassifier>

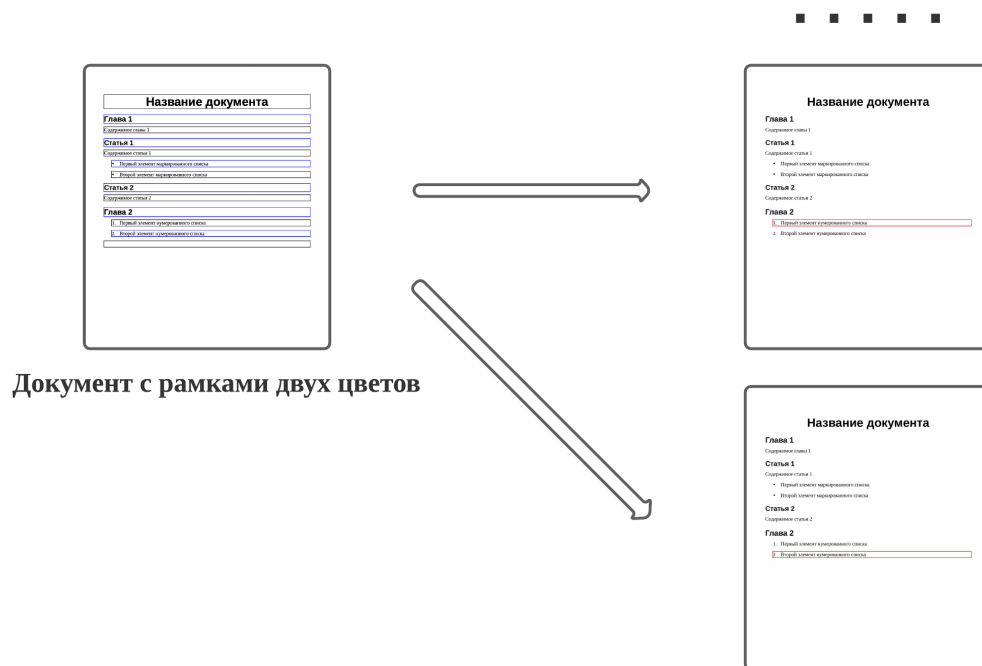
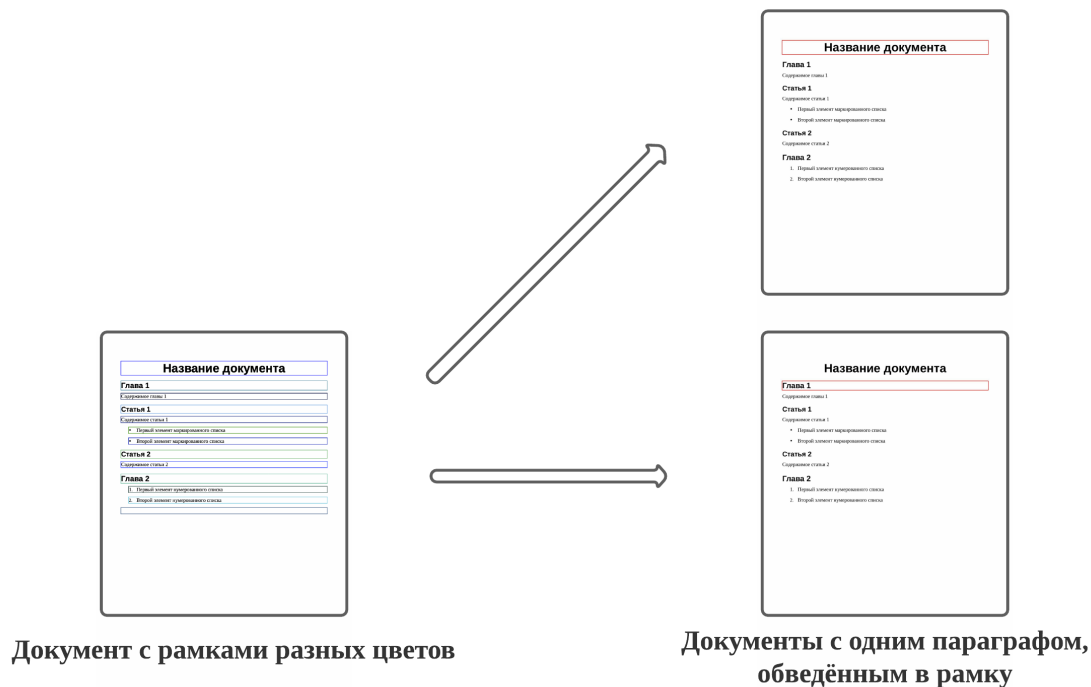


Рис. 6: Создание изображений для разметки из docx документа

1. Создание из docx документа двух pdf документов:

- документ с параграфами, обведенными рамками разных цветов;
- документ с параграфами, обведенными рамками двух цветов (цвета рамок чередуются для того, чтобы не происходило слияние рамок одного цвета).

Цвета рамок подбираются таким образом, чтобы вычитание из значений цветов первой группы цветов второй группы происходило корректно. Цвета описываются RGB представлением, получающимся из шестнадцатеричной записи чисел (вычитание цветов корректно, если число, их которого вычитают, было больше вычитаемого числа).

В docx документах параграфы обводились в рамку с помощью вставок специальных тегов в файл document.xml. Изменённые таким образом документы конвертируются в pdf документы. При создании документов с рамками, цвета рамок запоминаются.

2. Pdf документы конвертируются в изображения. Происходит параллельное чтение изображений двух документов. Находится разность между изображениями первого и второго документов. В итоге получаются изображения, содержащие только обводящие рамки.
3. Из изображений первого документа удаляются обводящие рамки с помощью маски, полученной из разницы изображений.
4. Зная значения цветов, использованных при рисовании рамок, можно на изображении, очищенном от рамок, последовательно рисовать рамки отдельно для каждого параграфа.
5. На выходе получается набор изображений, в каждом из которых один из параграфов обведён в рамку. С каждым изображением связан уникальный идентификатор параграфа, который был обведён в рамку, для установления соответствия между изображениями и параграфами документа.

Полученные таким образом изображения можно использовать для дальнейшей разметки.

4.5.2 Манифесты для классификаторов

Задача извлечения структуры из ТЗ документов разбивается на две подзадачи: классификация параграфов документа и сравнение пар параграфов документа. На основе решения этих двух подзадач строится решение глобальной задачи.

Типы для классификации параграфов ТЗ документов описаны в секции 4.1. В приложении 1 описаны указания по разметке документов и аналогично выделены 5 типов параграфов: титульный лист (title), оглавление (toc), структурные элементы (part), элементы списка (item) и текст (raw_text).

Сравнение параграфов можно рассматривать как классификацию пар параграфов на 3 класса: больше (greater), меньше (less) или равно (equal). Подробное описание того, как сравнивать параграфы, находится в приложении 2.

4.5.3 Набор данных для разметки и процесс разметки

Набор данных представляет собой набор из 22 документов, скачанных с сайта государственных закупок¹⁰. После преобразования документов в изображения для разметки получилось 1405 изображения с одним параграфом, обведённым в рамку. Документы были размечены (определялся тип параграфов) четырьмя аннотаторами с помощью системы разметки, разработанной в ИСП РАН¹¹.

Получившееся распределение параграфов по типам представлено на рисунке 7.

Аналогичным образом создавались изображения для сравнения пар параграфов. Для каждого параграфа, обведенного в рамку, обводился в рамку второй параграф, следующий после (не обязательно непосредственно после). Для каждого первого параграфа по отдельности обводились 4 параграфа, следующие за ним (если таковые имелись). Таким образом, для каждого параграфа документа создавалось по 4 изображения (или менее, если параграф являлся последним на странице). В итоговом наборе данных получилось 4438 изображений, размеченных аналогично предыдущему набору.

Получившееся распределение типов пар параграфов представлено на рисунке 8.

¹⁰<https://zakupki.gov.ru>

¹¹<https://github.com/dronperminov/ImageClassifier>



Рис. 7: Распределение параграфов по типам

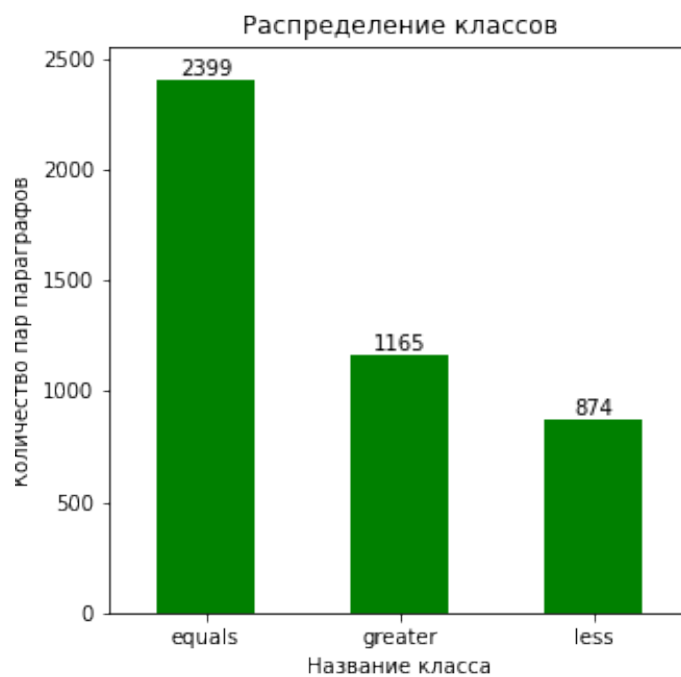


Рис. 8: Распределение типов пар параграфов

4.6 Оценка качества разработанного метода извлечения иерархической структуры

С помощью разработанных методов определения типов параграфов и сравнения параграфов было реализовано построение деревьев документов типа «Техническое задание».

Для оценки качества построения деревьев технических заданий была выбрана метрика Робинсона-Фулдса [15, 16]. Данная метрика интерпретируется как количество операций удаления и добавления рёбер, необходимое для преобразования одного дерева с типизированными узлами в другое. В оценке рассматривалась нормированная метрика, то есть количество применённых операций преобразования на общее количество операций преобразования. Таким образом, метрика может принимать значения от 0 до 1, при этом близость к нулю означает сходство деревьев.

Для измерения расстояния между правильным деревом документа и построенным были размечены 5 технических заданий. После сравнения размеченных деревьев и деревьев, полученных автоматически, усреднённое значение нормированной метрики Робинсона-Фулдса оказалось равным 0.073. Это показывает малое относительное число преобразований для превращения одного дерева документа в другое.

Несмотря на результаты, полученные на документах типа «Техническое задание», данный метод построения дерева может неправильно работать на документах других типов. Такой эффект возникает в силу специфики структуры технических заданий. Тем не менее, при наличии классификатора типов параграфов и компаратора пар параграфов для другого типа документа, по такой же схеме можно извлекать структуру из документов с другой моделью.

4.7 Реализация метода извлечения структуры

Так как для решения задачи необходимо уметь работать с данными и применять алгоритмы машинного обучения, для реализации всех методов логично выбрать язык программирования python. Для данной задачи увеличение скорости обработки документов не является ключевой целью, поэтому основным недостатком программ, написанных на python (скорость работы) можно пренебречь. Одной из причин использования python является наличие библиотек с реализацией алгоритмов машинного обучения, таких как

sklearn¹² и xgboost¹³. Кроме того, обработку docx документов необходимо встроить в проект dedoc¹⁴, написанный на python.

Методы для классификации параграфов документа, сравнения пар строк и построения дерева документа реализованы в виде набора классов. Классификатор типов параграфов и классификатор пар параграфов схожи между собой, поэтому для их реализации написаны абстрактные классы, методы которых конкретизируются в обоих классификаторах.

Реализованы следующие абстрактные классы:

1. AbstractFeatureExtractor – класс, предназначенный для превращения списка параграфов с метаданными в матрицу признаков. Основной абстрактный метод, реализуемый в классах наследниках, – это метод transform, который преобразует список списков словарей (словарь содержит текст и метаданные параграфа) в матрицу объект-признак. Кроме того, есть вспомогательные методы для добавления признаков предыдущих и следующих параграфов, извлечения определённых метаданных из словаря для параграфа и прочие вспомогательные методы.
2. AbstractLineTypeClassifier – непосредственно классификатор с абстрактным методом predict, который преобразует список словарей параграфов в список словарей с добавленными метками для каждого параграфа. Также у класса есть методы для сохранения классификатора в памяти и загрузки из памяти.
3. AbstractClassifierTrainer – класс, предназначенный для тренировки классификатора, содержит метод fit, в котором должна проводиться кросс-валидация и итоговое обучение классификатора и его сохранение. Помимо этого есть вспомогательные методы для сохранения ошибок классификатора и отрисовки матрицы ошибок.

Далее описаны модули с реализацией классификаторов и построением логической структуры:

1. tz_classifier – содержит реализацию извлекателя признаков для классификатора

¹²<https://scikit-learn.org>

¹³https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.

sklearn

¹⁴<https://github.com/ispras/dedoc>

параграфов, сам классификатор параграфов и класс для обучения классификатора.

2. `pair_classifier` – устроен аналогично классификатору типов параграфов, но предназначен для классификации пар параграфов.
3. `tree_constructor` – содержит класс для построения дерева документа из списка параграфов, функцию визуализации дерева и класс для построения правильного дерева документа (использовался при разметке документов для тестовой выборки).

4.7.1 Классификатор параграфов (`tz_classifier`)

Модуль `tz_classifier` содержит несколько файлов со следующими классами:

1. `TzTextFeatures` – класс-наследник абстрактного класса `AbstractFeatureExtractor`, реализующий метод `transform`, в котором соединяются признаки для параграфа (метод `_one_line_features`) с признаками списка (метод `_list_features`), заголовка (метод `__find_tz`) и оглавления (метод `__find_toc`).
2. `TzLineTypeClassifier` – класс-наследник абстрактного класса `AbstractLineTypeClassifier`, который реализует метод `predict`. Вспомогательный метод `__get_predictions` производит постобработку ответов классификатора и используется методом `predict`.
3. `TzClassifierTrainer` – класс-наследник абстрактного класса `AbstractClassifierTrainer`, предназначенный для проведения кросс-валидации и сохранения обученного классификатора. Метод `__get_data` загружает данные и преобразует их, разбивая по группам, для того чтобы параграфы одного документа оказались либо полностью в тренировочном наборе, либо в валидационном. Метод `_cross_val` нужен для кросс-валидации и сохранения ошибок в процессе кросс-валидации, метод `fit` вызывает метод `_cross_val` и обучает итоговый классификатор, если это необходимо.

4.7.2 Классификатор пар параграфов (`pair_classifier`)

Модуль `pair_classifier` содержит несколько файлов со следующими классами:

1. `PairFeaturesExtractor` – класс-наследник абстрактного класса `AbstractFeatureExtractor`, который реализует метод `transform`. Метод `transform` преобразует список пар параграфов (каждая пара – список из двух словарей параграфов) в матрицу объект-признак с помощью метода `_get_pair_features`. Также класс содержит несколько вспомогательных методов, основной из которых – `__get_feature_difference_for_pair` нужен для вычисления разницы признаков для пары параграфов.
2. `PairClassifier` – класс-наследник абстрактного класса `AbstractLineTypeClassifier`, реализующий метод `predict`. Кроме того, есть метод `compare`, который для пары параграфов возвращает их метку (это удобно использовать при построении дерева).
3. `PairClassifierTrainer` – класс-наследник абстрактного класса `AbstractClassifierTrainer`, предназначенный для проведения кросс-валидации и сохранения обученного классификатора. Методы `__get_data`, `_cross_val` и `fit` выполняют функции, аналогичные тем же методам у `TzClassifierTrainer`.

4.7.3 Конструктор дерева документа (`tree_constructor`)

Листинг 1: Структура узла извлекаемого дерева

```
{ "type": "some_type",  
  "data": {},  
  "children": [],  
  "parent": {} }
```

В соответствии с описанным в секции 4.2 алгоритмом на основе реализованного компаратора строк написан код для построения дерева документа. Структура документа извлекается в виде словаря, который легко перевести в json формат для сохранения данных (см. листинг 1).

В данной структуре «type» означает тип параграфа (например, один из типов, описанных для технических заданий), если интересна структура дерева без типов, значением этого поля будет пустая строка. Поле «data» содержит текст параграфа со всеми извлечёнными аннотациями. Поле «children» содержит список словарей узлов – это

параграфы, вложенные в данный. Поле «parent» содержит словарь для узла предка текущего параграфа. Узел корня дерева не содержит полей «data» и «parent» и имеет тип «root».

Помимо построения дерева документа реализована его визуализация с помощью библиотеки `treelib`¹⁵. Пример визуализации дерева документа без типов показан на рис. 9.

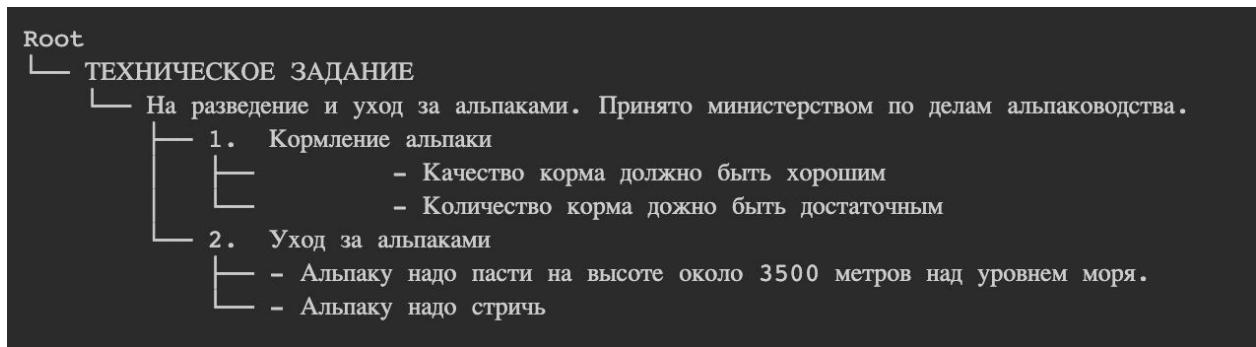


Рис. 9: Пример визуализации дерева документа

Для документов типа «Техническое задание» реализовано отдельное построение дерева, в котором все параграфы, относящиеся к заголовку, становятся вложенными в наибольший параграф заголовка, аналогично с параграфами содержания. Это сделано для того, чтобы остальные параграфы документа не оказывались вложенными в содержание, которое находится в начале документа. Для документов других типов можно реализовать другое построение дерева в соответствии с конкретной моделью документа.

Визуализация типизированного дерева для технического задания показана на рис. 10.

Реализация построения дерева составляет модуль `tree_constructor`. Модуль `tree_constructor` содержит несколько файлов со следующими классами:

1. `DocumentTreeConstructor` – класс для преобразования списка параграфов документа в дерево специального вида (см. листинг 1). Метод `construct_tree` возвращает словарь с полученным деревом. Кроме того, реализован дополнительный класс `TzDocumentTreeConstructor`, аналогичный классу `DocumentTreeConstructor`, но отдельно рассматривающий заголовок и содержание и строящий структуру, описанную для технических заданий (см. секцию 4.1).

¹⁵<https://treelib.readthedocs.io/en/latest/>

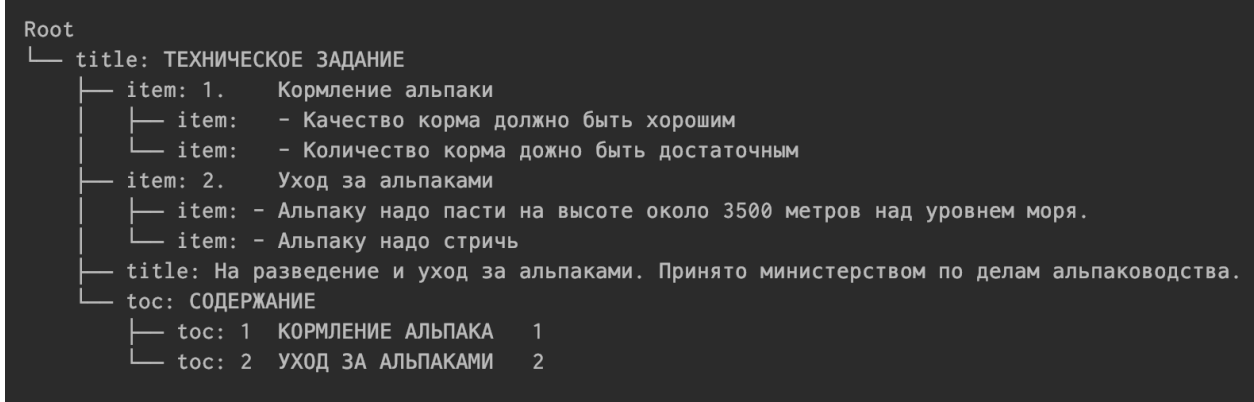


Рис. 10: Пример визуализации типизированного дерева для технического задания

2. `TestTreeConstructor` – строит дерево аналогично `DocumentTreeConstructor`, но вместо вызова компаратора параграфов просит пользователя сравнить параграфы вручную. Возвращает список результатов разметки, на основе которого строятся тестовые деревья для сравнения с построенными автоматически.
3. Файл `tree_visualization.py` содержит функцию `visualize_tree`, которая визуализирует дерево документа, полученное с помощью `DocumentTreeConstructor` (см. рисунок 9).

Весь код реализованных методов содержится в модуле `classifiers` в репозитории на github.com¹⁶.

4.8 Сравнение с существующим методом

В главе 3 описан метод классификации параграфов `docx` на 3 типа: секция, подсекция, содержимое [11]. Данный метод предназначен для классификации параграфов `docx` документов на типы, отличающиеся от типов параграфов в ТЗ документах. Однако можно применить существующий метод для классификации параграфов на типы параграфов ТЗ и рассмотреть результат работы данного метода на размеченном наборе данных их ТЗ документов.

Так как кода реализованного метода нет в свободном доступе, данный существующий метод был реализован на языке `python` аналогично классификатору параграфов

¹⁶<https://github.com/NastyBoget/DOCXParser>

ТЗ. В качестве признаков использованы описанные в статье признаки: размер шрифта (нормализованный), отступ (есть или нет), точка в конце, несколько предложений, выравнивание (4 типа), жирность, нумерация, подчеркивание. Для классификации использован алгоритм Случайный лес (RandomForestClassifier [17]).

Существующий метод не предназначался для выделения содержания в документах, поэтому при сравнении методов производилась классификация на 4 класса: название, часть, список, текст. То есть из тренировочных и валидационных данных убирались параграфы с меткой «содержание». В результате кросс-валидации получились усреднённые значения достоверности и F-меры для реализованного классификатора параграфов ТЗ и существующего метода, представленные в таблице 5.

Метод	Accuracy	F1-measure (macro)
Реализованный метод	0.95212	0.93117
Реализованный метод с постобработкой	0.96847	0.96384
Существующий метод	0.91314	0.86364

Таблица 5: Сравнение результатов работы реализованного и существующего методов

4.9 Выводы

Исследована логическая структура технических заданий и предложена структура следующего вида: дерево произвольной глубины, в котором каждому узлу соответствует параграф документа с типом (заголовок, содержание, часть, элемент списка или текст).

Разработана и реализована система для автоматической обработки технических заданий в формате docx, позволяющая делать следующее:

- Получать для документа (технического задания) список словарей параграфов с метаданными и метками 5 типов (заголовок, содержание, часть, элемент списка и текст). Достоверность (ассигасу) определения типа на кросс-валидации равна 0.96.
- Получать для документов структуру в виде дерева в соответствии с предложенной логической структурой. Дерево можно строить как с типизированными узлами,

определяя тип с помощью классификатора типов параграфов, так и с нетипизированными узлами. Усредненное значение расстояния Робинсона-Фулдса на тестовых данных оказалось равным 0.073.

- Визуализировать получившиеся деревья для документов (см. рисунок 9).

5 Описание практической части

Согласно главе 1 необходимо реализовать извлечение текста и метаданных из docx документов на языке программирования python. В секции 5.1 описаны существующие библиотеки по работе с docx документами, возможностей которых недостаточно для реализации метода извлечения структуры. Поэтому следует изучить устройство docx формата и реализовать обработку документов и извлечение всей описанной выше информации.

5.1 Существующие библиотеки по работе с форматом docx

В главе 4 обосновывалось использование языка python для реализации обработчика docx документов. Существует большое количество библиотек на языке программирования python для работы с docx документами. Рассмотрим некоторые из них с точки зрения выделения метаданных, необходимых для решения нашей задачи. Для реализации методов необходимо извлекать весь текст параграфов документа и метаданные, описанные в главе 4. Необходимо исследовать возможности библиотек по выделению данной информации из документов.

5.1.1 python-docx

Python-docx¹⁷ – наиболее известная библиотека по работе с docx документами для языка программирования python. Она предоставляет возможности для создания, чтения и редактирования документов. При чтении документа создается класс Document, хранящий всю информацию, которую удалось извлечь. Текст документа и некоторые метаданные хранятся в последовательности параграфов (класс Paragraph) из атрибута Document.paragraphs.

Данная библиотека имеет ряд недостатков:

- Не извлекается текст нумерации для элементов списков. Эта информация очень важна для анализа структуры документов, она является ключевой при выделении элементов списков из документов.

¹⁷<https://python-docx.readthedocs.io/en/latest/>

- Не всегда извлекаются все метаданные параграфов. Так, при переопределении стиля части некоторого параграфа (например, переопределение жирности шрифта) эта информация не будет извлечена, то есть часть метаданных для параграфа будет утеряна. При этом, метаданные из стилей извлечь можно.

Информация о жирности, подчеркивании, курсиве и размере шрифта содержит в себе важные признаки для дальнейшего обучения классификаторов, однако данная библиотека может игнорировать эту информацию.

- Помимо недостающей функциональности, библиотека предоставляет возможности, которые не нужны в рамках текущей задачи. Для извлечения структуры достаточно уметь читать файл без возможности его изменения.

5.1.2 docx2python

Docx2python¹⁸ – библиотека для чтения docx документов и их представления в html формате. В отличие от python-docx, помимо основного текста документа docx2python также извлекает текст нумерации. Данная библиотека предоставляет возможность прочитать только текст документа, а также получить html представление документа, в котором может присутствовать информация о жирности, курсиве, подчеркивании, размере шрифта, измененная явно в документе.

Данная библиотека имеет следующие недостатки:

- Нумерация извлекается, однако текст нумерации однотипный: цифра со скобкой для нумерованных списков и двойное тире для маркированных списков. Таким образом, точный текст нумерации получить нельзя.
- Информация о жирности, курсиве и т. д. извлекается не всегда: из стилей метаданные не извлекаются. Поэтому большая доля информации о тексте утрачивается.
- Нет информации о выравнивании, отступах в документе.

¹⁸<https://pypi.org/project/docx2python/>

5.1.3 pydocx

Pydocx¹⁹ – библиотека для конвертации docx документов в html формат определенного вида. Реализовано выделение жирности, курсива, подчеркивания и выравнивания. Кроме того, есть возможность определения заголовков различных уровней, нумерованных и маркированных списков. Однако выбранный формат не всегда подходит для хранения всех необходимых метаданных для параграфов. Среди основных недостатков можно выделить следующие:

- Выбранный в библиотеке выходной формат не предоставляет информацию об отступах и размере шрифта. Эта информация может существенно улучшить результат работы классификаторов.
- Недостаточная информация о списках. Выходной формат позволяет выявлять списки и их тип (нумерованный, маркированный), однако текст нумерации теряется. Таким образом, нельзя получить точный текст нумерации документа.

5.1.4 Выводы

По результатам анализа возможностей рассмотренных библиотек можно сделать вывод, что каждая из библиотек позволяет извлечь из docx документов некоторую информацию, которая частично удовлетворяет требованиям нашей задачи. Стоит особо выделить ограниченные возможности библиотек по выделению текста нумерации для автоматических списков. Однако извлечение более полной информации из документов может существенно повысить качество работы методов извлечения структуры. Кроме того, возможности, предоставляемые библиотеками, трудно расширить новыми, если таковые понадобятся в дальнейшем.

5.2 Составляющие docx-файла

Документация по docx формату содержится в специально разработанном стандарте [13]. Docx-файл — это zip-архив который физически содержит 2 типа файлов:

- xml файлы с расширениями xml и rels;

¹⁹<https://github.com/CenterForOpenScience/pydocx>

- медиа файлы (изображения и т.п.).

Логически файл содержит 3 вида элементов:

- Типы (Content Types) — список типов медиа файлов (например png) встречающихся в документе и типов частей документов (например документ, верхний колонтитул);
- Части (Parts) — отдельные части документа, то есть непосредственно его содержимое (например document.xml, footer1.xml, header1.xml, comments.xml, endnotes.xml), сюда входят как xml документы, так и медиа файлы;
- Связи (Relationships) идентифицируют части документа для ссылок (например связь между разделом документа и колонтитулом), а также тут определены внешние части (например гиперссылки).

В рамках поставленной задачи требуется выделить текстовые данные из документов, а также некоторые метаданные, то есть свойства отдельных частей текста (например, жирность и размер шрифта). Список необходимых метаданных описан в секции 4. Далее слова «метаданные» и «свойства» для параграфов имеют одинаковый смысл.

Для выделения текста и метаданных необходимо обработать части документа (Parts), связанные с его содержимым. Ограничим содержимое документа текстом его параграфов. Исключим из рассмотрения таблицы и различные вложения типа картинок и документов других форматов.

Содержимое для анализа содержится в трёх файлах с расширением xml:

- *document.xml* — содержит текст документа и некоторые свойства параграфов (описание метаданных, ссылки на стили и нумерацию);
- *styles.xml* — содержит описание стилей, используемых в документе;
- *numbering.xml* — содержит информацию о типах нумерованных и маркированных списков, используемых в документе, а так же о стилях списков.

Для разбора этих файлов необходимо знать основные понятия, использующиеся в данных файлах и в документации формата.

5.3 Некоторые понятия, используемые в документации формата

Язык xml является языком разметки, то есть содержимое документа размечается специальными конструкциями – тегами. Тег оборачивается в угловые скобки и имеет название. Теги вкладываются друг в друга, образуя иерархию. Помимо названия, теги могут иметь атрибуты – названия полей, которым присвоены строковые значения. В дальнейшем понятия «тег» и «атрибут» будут использоваться в этом смысле.

5.3.1 Тело документа (body)

Тело документа – это основное текстовое содержимое документа, представляющее из себя последовательность параграфов, таблиц, встроенных графиков и прочих элементов. Тело документа располагается в файле document.xml внутри тега body. Для каждого параграфа по отдельности описаны все его свойства, тело документа выполняет лишь роль перечислителя параграфов в определённом порядке. Как было сказано выше, мы ограничиваемся рассмотрением только параграфов документа, исключая из рассмотрения все остальные элементы, которые могут встретиться в документе.

5.3.2 Параграф (paragraph)

Параграф – это основная структурная единица документа. Документ делится на параграфы переносом строки, набранным в текстовом редакторе. Параграф соответствует тегу p документа.

У параграфа может присутствовать набор свойств, которые описывают отдельный блок текста. Выделяются так называемые «прямые свойства», которые применяются к параграфу напрямую и прописаны в document.xml, а так же «косвенные свойства», описанные в стилях из styles.xml и применяемые в силу того, что в параграфе есть ссылка на определенный стиль. Кроме того, в свойствах параграфа (как в прямых, так и в косвенных) может содержаться информация о том, что параграф является элементом списка. Свойства параграфа соответствуют тегу pPr документации (этот тег вложен в тег p).

Теги, соответствующие некоторым свойствам параграфа:

- ind (indentation) – отступ от края страницы, атрибут left содержит значение отступа от левого края страницы в двадцатых пункта (пункт – это 1/72 дюйма);

- js (justification) – выравнивание, значением атрибута val может быть left (по левому краю), right (по правому краю), center (по центру), both (по обоим краям);
- sz (size) – размер шрифта, значение атрибута val содержит размер шрифта в половинах пункта;
- numPr (numbering properties) – индикатор того, что параграф является элементом нумерованного или маркированного списка. Атрибут numId содержит уникальный идентификатор списка, элементом которого является данный параграф, атрибут lvl содержит уровень вложенности списка (нумерация с нуля);
- pStyle (paragraph style) – стиль параграфа, значением атрибута val является идентификатор стиля из файла styles.xml.

Параграф, в свою очередь, состоит из списка текстовых элементов с общими свойствами (run). Каждый текстовый элемент содержит свой отдельно описанный набор свойств.

5.3.3 Текстовый элемент (run)

Текстовый элемент (run) – элемент, составляющий текстовый регион с набором общих свойств (работает на уровне символов). Например, пусть одна часть параграфа написана жирным шрифтом, а другая – обычным. Если остальные свойства текста совпадают, параграф будет состоять из двух текстовых элементов. Помимо описания свойств текста, элемент содержит в себе непосредственно сам текст. Текстовый элемент соответствует тегу r.

У текстового элемента, так же как и у параграфа, описывается набор свойств, как прямых, так и косвенных (то есть может быть ссылка на стиль из styles.xml). Свойства текстового элемента соответствуют тегу rPr документации.

Теги, соответствующие некоторым свойствам текстового элемента:

- b (bold) – жирный шрифт, значение атрибута val может быть 0 (или False, false) – шрифт нежирный, либо 1 (или True, true) – шрифт жирный. Если атрибутов нет, но тег присутствует, то шрифт жирный, если тега нет, то шрифт нежирный;
- i (italic) – курсив, описание атрибутов то же, что и у bold;

- `u` (`underlined`) – подчеркивание, значение атрибута `val` может быть `none` – шрифт не подчеркнут, либо указан вид подчеркивания (например, `double`). Если атрибутов нет, но тег присутствует, то шрифт подчеркнутый, если тега нет, то шрифт не подчеркнутый;
- `rStyle` (`run style`) – стиль элемента, значением атрибута `val` является идентификатор стиля из файла `styles.xml`;
- `sz` (`size`) – размер шрифта (аналогичен тому же свойству у параграфа);
- `t` (`text`) – текст элемента, отображающийся в документе (текст находится внутри тега);
- `caps` – представление текста заглавными буквами, значение атрибута `val` может быть 0 (или `False`, `false`) – не отображать текст заглавными буквами, либо 1 (или `True`, `true`) – отображать. Если атрибутов нет, но тег присутствует, то отображать текст заглавными буквами, если тега нет, то не отображать;
- `tab` (табуляция), `br` (перенос строки), `cr` (возврат каретки), `sym` – специальные символы, которые могут встретиться в тексте элемента, у тега `sym` значением атрибута `char` является шестнадцатеричный код символа.

Некоторые свойства могут быть описаны как для параграфов, так и для текстовых элементов. В рамках данной работы, общим свойством является размер шрифта. В таком случае, если свойство не описано в свойствах элемента, рассматривается значение этого свойства для параграфа, иначе свойство элемента перекрывает свойство параграфа.

5.3.4 Стиль (`style`)

Стиль – основная структурная единица файла `styles.xml`, соответствующая тегу `style`. Стиль содержит в себе описание свойств конкретного элемента в зависимости от типа стиля, то есть тип стиля соответствует типу элемента (таблица, параграф, нумерация, символ). Если стиль описывает свойства параграфа, он также может включать в себя индикатор того, что параграф является элементом списка.

В файле `styles.xml` описывается набор стилей, примененных к различным элементам документа, и отношения наследования между ними. Если один из стилей является наследником другого, то все не перекрытые свойства стиля-предка становятся свойствами стиля-наследника. Каждый стиль имеет уникальный идентификатор и тип, по этой паре значений можно ссылаться на любой стиль документа. Уникальный идентификатор стиля – это значение атрибута `styleId` тега `style`, тип стиля – значение атрибута `type`. Кроме того, если у стиля есть атрибут `default`, значение которого равно 1, то данный стиль является стилем по умолчанию среди всех стилей данного типа.

Свойства текста для документа определяются в соответствии с диаграммой, показанной на рис. 11. Таким образом, к тексту сначала применяются настройки документа по умолчанию, затем свойства параграфа, нумерации и текстового элемента, описанные в стилях, и, наконец, изменяются «прямые свойства», описанные непосредственно в `document.xml`.



Рис. 11: Порядок применения настроек свойств

Помимо тегов, описывающих свойства параграфа и текстового элемента (`pPr` и `rPr`), в стилях выделяются следующие теги:

- `basedOn` – наследование стилей (параграфы и символы наследуют свойства параграфов и символов соответственно, нумерация не наследуется), значение атрибута `val` содержит уникальный идентификатор стиля-родителя;
- `docDefaults` – настройки стиля по умолчанию (данный тег вложен в тег `styles`

наравне с тегами `style` и устроен аналогично им).

В стилях параграфов может встречаться индикатор того, что параграф является элементом списка. В таком случае в стиле не будет информации об уровне вложенности списка (нет атрибута `lvl`), однако в описании списка в `numbering.xml` на одном из уровней будет присутствовать ссылка на стиль.

Стили для нумерации содержат лишь ссылку на описание списка в `numbering.xml`.

5.3.5 Абстрактная и непосредственная нумерация (`abstract numbering and numbering`)

В файле `numbering.xml` содержится информация обо всех типах списков, используемых в документе и их настройках. В документации выделяется два основных понятия: абстрактная нумерация (описание свойств абстрактного списка) и непосредственно нумерация (описание конкретного списка). Абстрактный список описывает свойства списка и может быть общим для нескольких списков, использующих описание свойств.

Абстрактная нумерация (`abstract numbering`) – описание свойств абстрактного списка (соответствует тегу `abstractNum`), этому типу присваивается уникальный идентификатор (атрибут `abstractNumId`), который может использоваться в конкретных реализациях списков. Абстрактная нумерация не может использоваться нигде помимо непосредственной нумерации, наследующей её свойства. Абстрактная нумерация описывает некоторые свойства, общие для всех уровней списка, и свойства, присущие каждому уровню по отдельности. Абстрактные списки могут наследовать свойства других абстрактных списков.

Непосредственная нумерация или нумерация (`numbering`) – описание конкретных списков документа. Каждый список документа соответствует одному (или нескольким в случае разнородного по стилям списка) списку нумерации (соответствует тегу `num`). Список нумерации имеет уникальный идентификатор (атрибут `numId`), с помощью которого в `document.xml` и `styles.xml` можно ссылаться на данный список. Кроме того, каждый список нумерации с помощью `abstractNumId` ссылается на абстрактный список, свойства которого наследуются и могут перегружаться.

Таким образом, файл `numbering.xml` содержит последовательность из конкретных списков (`num`), каждый из которых ссылается на абстрактный список (`abstractNum`) (абстрактные списки так же могут ссылаться на другие абстрактные списки) и может

менять некоторые свойства отнаследованного абстрактного списка. В файле `document.xml` параграфы ссылаются на `numId` списков напрямую или через стили (непосредственно на списки `num`, которые наследуются от абстрактных списков `abstractNum`), тем самым становясь элементами списка. Нумерация списка увеличивается в соответствии с появлением новых элементов одного и того же абстрактного списка, то есть нумерация сохраняется в пределах абстрактного списка до тех пор, пока он не будет прерван другим абстрактным списком (есть исключения, описанные ниже).

Некоторые теги, соответствующие свойствам абстрактного списка (`abstractNum`):

- `numStyleLink` – вместо описания свойств абстрактный список может хранить ссылку на другой список, свойства которого будут скопированы в текущий список. Значение атрибута `val` данного тега равно значению атрибута тега `styleLink` того списка, свойства которого будут отнаследованы;
- `styleLink` – индикатор того, что данный абстрактный список описывает свойства, на которые могут ссылаться, атрибут `val` содержит имя, посредством которого можно ссылаться на эти свойства;
- `restartNumberingAfterBreak` – это атрибут тега `abstractNum` для более современных версий приложений, если его значение равно 0, то нумерация не начинается заново, даже если после данного абстрактного списка встречался другой абстрактный список;
- `lvl` – содержит информацию о свойствах конкретного уровня списка, значение атрибута `ilvl` равно номеру уровня.

Некоторые теги, соответствующие свойствам конкретного списка (`num`):

- `abstractNumId` – значение атрибута `val` содержит уникальный идентификатор абстрактного списка, свойства которого наследуются;
- `lvlOverride` – используется для перегрузки свойств некоторых уровней списка. Данный тег содержит список уровней (тегов `lvl`), каждый из которых перекрывает свойства уровней, описанных в абстрактном списке.

Теги, соответствующие свойствам одного из уровней списка (теги, вложенные в тег `lvl`):

- `lvlText` – текстовое представление нумерации списка. В значении атрибута `val` указана строка специального вида: если в ней встречается символ `%` с последующей цифрой (цифра означает номер уровня + 1), то в итоговом тексте нумерации вместо `%` и цифры вставляется текущее значение счётчика нумерации списка для данного уровня;
- `numFmt` – формат, в котором представляется список на конкретном уровне, значения формата представлены атрибутом `val`, например, `decimal` (десятичное число), `lowerRoman` (римские цифры в нижнем регистре) и т. д.;
- `isLgl` – если этот тег присутствует, то необходимо игнорировать тег `numFmt` для всех уровней и использовать нумерацию десятичными цифрами;
- `start` – начальное значение нумерации (для первого элемента списка или для списка, который начался сначала из-за `lvlRestart`). Значение атрибута `val` – десятичное число, показывающее номер, с которого начинается отсчет, если тега `start` нет, то его значение устанавливается в 0;
- `lvlRestart` – если значение атрибута `val` равно 0 и предыдущий элемент данного списка находился выше по иерархии, то не начинать нумерацию сначала, иначе нумеровать сначала (в том числе, если тег отсутствует);
- `suff` – содержимое между текстом нумерации и остальным текстом параграфа, значением атрибута `val` может быть `nothing` (пустая строка), `space` (пробел), `tab` (табуляция);
- `pStyle` – в атрибуте `val` содержит для конкретного уровня уникальный идентификатор стиля из `styles.xml` (этот стиль содержит индикатор нумерации без указания уровня). Если параграф в `document.xml` ссылается на этот стиль, то этот параграф считается пронумерованным и уровнем вложенности будет тот уровень, в котором присутствует тег `pStyle` с указанием данного стиля;
- `pPr` – свойства пронумерованного параграфа для данного уровня аналогичные свойствам параграфа, описанным выше. Если для пронумерованного параграфа в `document.xml` прописаны свойства, они перекроют данные свойства нумерации;

- `rPr` – свойства текстового элемента, которые применяются только к тексту нумерации;
- `startOverride` – данный тег может присутствовать внутри `lvlOverride` и отвечает за сбрасывание нумерации того абстрактного списка, наследником которого является список, содержащий этот тег. Значение атрибута `val` показывает новое начальное значение нумерации.

5.4 Реализация обработчика docx документов

Как указано в главе 4, для решения задачи необходимо реализовать извлечение из docx документов текста и следующих метаданных:

- размер шрифта;
- отступ от края страницы;
- выравнивание;
- жирность шрифта;
- курсив;
- подчеркивание.

Из описания формата в секции 5.2 следует, что для выделения всего текста из документов необходимо анализировать файл `document.xml` с основным текстом, а также `numbering.xml` с текстом нумерации для автоматических списков. Для выделения указанных метаданных помимо этих двух файлов необходимо анализировать файл `styles.xml` со стилями.

Для обработки docx файлов был реализован набор классов на языке python. Общее устройство обработчика docx файлов выглядит следующим образом (описано содержимое модуля `docx_parser`):

- `document_parser` – основной модуль с обработчиком файлов;
- `data_structures` – набор вспомогательных структур, таких как параграф, текстовый элемент (run по документации) и свойства параграфа;

- `extractors` – набор обработчиков конкретных видов данных (например, стилей и нумерации), в который входят следующие модули: `properties_extractor`, `styles_extractor`, `numbering_extractor`;
- `properties_extractor` – модуль для выделения из xml-параграфов различных свойств;
- `styles_extractor` – модуль для анализа стилей и применения стилей к параграфам;
- `numbering_extractor` – модуль для извлечения текстового содержимого нумерации списков.

Модуль `document_parser` содержит класс `DOCXParser` со следующими методами:

- `can_parse` – проверяет, можно ли обработать файл с заданным именем;
- `parse` – обрабатывает файл, результат сохраняется внутри класса;
- `get_lines_with_meta` – возвращает список строк документа с необходимыми метаданными.

Модуль `data_structures` содержит несколько модулей:

- `base_props` – содержит базовый класс `BaseProperties` для `Run` и `Paragraph`, который хранит в себе общие для `Paragraph` и `Run` свойства: отступ, выравнивание, размер шрифта, жирность, курсив, подчеркивание. У параграфов и у их частей могут присутствовать и отсутствовать некоторые общие свойства, которые применяются по иерархии, поэтому удобно использовать общий базовый класс.
- `paragraph` – соответствует параграфу `docx`, содержит класс `Paragraph`, который помимо свойств `BaseProperties` хранит список текстовых элементов (`run`), информацию о стиле и уровне вложенности списка (если есть). Метод `parse` формирует свойства параграфа в соответствии с иерархией применяемых стилей и сохраняет их внутри класса.
- `run` – соответствует текстовому элементу (`run`) в `docx`, содержит класс `Run`, который помимо свойств `BaseProperties` хранит текст части параграфа. Метод `get_text` формирует текст элемента на основе `xml` и сохраняет его внутри класса.

- `paragraph_info` – содержит класс `ParagraphInfo`, который преобразует свойства параграфа в аннотации – для параграфа формируется список кортежей с названием аннотации, ее значением и координатами начала и конца применения аннотации к тексту параграфа. Метод `get_info` – формирует и возвращает для параграфа словарь с его текстом и списком аннотаций.

Модуль `properties_extractor` содержит набор функций, которые на основе `Run` или `Paragraph` и xml-содержимого обновляют свойства параграфа или текстового элемента:

- `change_paragraph_properties` – изменяет отступ, размер шрифта и выравнивание параграфа;
- `change_run_properties` – изменяет жирность, курсив, подчеркивание, размер шрифта;
- `change_indent` – изменяет отступ;
- `change_size` – изменяет размер шрифта;
- `change_js` – изменяет выравнивание.

Модуль `styles_extractor` содержит класс `StylesExtractor`, в конструкторе которого сохраняются xml-деревья с настройками стилей по умолчанию; метод `parse` изменяет свойства `Paragraph` или `Run` в соответствии с иерархией стилей и с тем стилем, `styleId` которого передан, также может добавиться текстовый элемент с нумерацией.

Модуль `numbering_extractor` содержит следующие классы:

- класс `AbstractNum` соответствует `abstractNum` в docx, описывает свойства типа списка. В конструкторе инициализируются свойства, общие для всех уровней и словарь свойств для каждого уровня; метод `parse` обрабатывает список уровней, заполняя свойства каждого уровня.
- класс `Num` соответствует `num` в docx, наследует все свойства одного из `AbstractNum` и перегружает некоторые из них, в конструкторе получает информацию о свойствах всех уровней списка. Метод `get_level_info` возвращает словарь со свойствами для конкретного уровня списка.

- класс `NumberingExtractor` строит соответствие между `Num` и `AbstractNum` и вычисляет текст и свойства нумерации списков. Метод `parse` формирует текстовый элемент с нумерацией и добавляет нумерацию к переданному в качестве параметра параграфу.

5.5 Тестирование обработчика docx документов

Для тестирования обработчика docx документов был создан набор файлов, содержимое которых позволяет покрыть основные функции, выполняемые обработчиком. Так как xml-содержимое документов может различаться в зависимости от операционной системы и приложения, в котором создается документ, документы создавались в трёх операционных системах (Windows, Linux, Mac OS) с помощью трех различных приложений (Microsoft Word, LibreOffice, Pages).

Проверялась следующая информация, выделенная из документов:

- Аннотации, явно задаваемые пользователем при создании документа: жирность, курсив, подчеркивание, размер шрифта, выравнивание, отступ, название стиля, написание текста заглавными буквами.
- Те же аннотации, заданные косвенно с помощью стилей.
- Текстовое содержимое нумерации списков. Генерировались автоматические списки со сложной многоуровневой структурой, с различными типами нумерации и чередующиеся списки разных типов. Анализировался полученный в результате анализа таких списков текст.
- Нумерация, заданная косвенно с помощью стилей.

Кроме того, проведено тестирование обработчика документов на наборе из 570 документов, скачанных с сайта государственных закупок²⁰. Это позволило увеличить разнообразие документов для проверки безошибочной работы обработчика.

Код обработчика docx документов встроен в открытый проект `dedoc`²¹, разработанный в ИСП РАН. Было измерено время работы обработчика документов на наборе из

²⁰<https://zakupki.gov.ru>

²¹<https://github.com/ispras/dedoc>

технических заданий (см. секцию 4.5). Были получены следующие результаты: средний документ обрабатывается за 2.6 секунды, средний размер документа 461477 байтов (8012 символов). Общее время обработки всех 22 документов составило 57 секунд.

5.6 Выводы

Реализована обработка документов в формате docx на языке python. Результатом обработки документа является список его параграфов с текстом (вместе с нумерацией автоматических списков) и метаданными (размер шрифта, отступ от края страницы, выравнивание, жирность шрифта, курсив, подчеркивание). Проведено тестирование работы реализованного обработчика. Код обработчика docx документов встроен в открытый проект dedoc²², разработанный в ИСП РАН.

²²<https://github.com/ispras/dedoc>

6 Заключение

В ходе работы была исследована структура технических заданий на автоматизированные системы. В результате изучения различных вариантов ТЗ и существующих подходов к автоматизированному анализу ТЗ было выявлено, что техническое задание имеет иерархическую структуру, т.е. состоит из вложенных друг в друга частей. Предложена иерархическая структура в виде дерева произвольной глубины, в котором каждому узлу соответствует параграф документа одного из следующих типов: заголовок, содержание, часть, элемент списка или текст. Разработан метод по извлечению предложенной иерархической логической структуры из технических заданий, представленных в формате docx.

Построение дерева логической структуры осуществляется на основе сравнения параграфов с помощью классификатора пар параграфов. Тип параграфов определяется классификатором параграфов.

Для проведения тестирования качества работы разработанного метода был составлен и размечен набор технических заданий из 22 документов. Набор данных находится в открытом доступе²³. Было размечено 1405 параграфов (для типа) и 4438 пар параграфов (для сравнения параграфов). Усреднённое значение достоверности (ассигасу) определения типа параграфа на кросс-валидации равно 0.96. Усреднённое значение достоверности (ассигасу) сравнения параграфов на кросс-валидации равно 0.98. После автоматического построения деревьев на тестовой выборке из 5 документов усреднённое значение расстояния Робинсона-Фулдса между построенными и размеченными деревьями оказалось равным 0.073. Показанные уровни качества достаточно высоки для использования предложенного метода.

Для решения поставленной задачи не было найдено существующих методов, однако был найден метод классификации параграфов docx документов любого домена. Поэтому проведено сравнение реализованного метода классификации параграфов технического задания с адаптацией существующего метода [11] к поставленной задаче. Существующий метод не предназначался для выделения содержания в документах, поэтому при сравнении методов производилась классификация на 4 класса (заголовок, часть, список и текст) без учёта содержания. Существующий метод был реализован в

²³<https://github.com/NastyBoget/DOCXParser>

соответствии с его описанием и изменен для классификации параграфов на 4 класса, при сравнении оказалось, что существующий метод справляется с поставленной задачей хуже, чем метод, предложенный в данной работе.

Реализованный метод по извлечению иерархической логической структуры из технических заданий доступен в виде библиотеки²⁴, которая позволяет из технических заданий в формате docx получить дерево в виде словаря python.

В качестве вспомогательной задачи реализовано автоматическое извлечение текста и некоторых метаданных из документов в формате docx на языке python. В отличие от других библиотек языка python реализованный обработчик поддерживает корректное извлечение текста нумерации автоматических списков. Код обработчика docx документов встроен в проект dedoc²⁵.

²⁴<https://github.com/NastyBoget/DOCXParser>

²⁵<https://github.com/ispras/dedoc>

Список литературы

- [1] TRIE: End-to-End Text Reading and Information Extraction for Document Understanding / Peng Zhang, Yunlu Xu, Zhanzhan Cheng et al. // Proceedings of the 28th ACM International Conference on Multimedia. — 2020. — Pp. 1413–1422.
- [2] Roh, Taeyeoun. Developing a Methodology of Structuring and Layering Technological Information in Patent Documents through Natural Language Processing / Taeyeoun Roh, Yujin Jeong, Byungun Yoon // *Sustainability*. — 2017. — Vol. 9, no. 11. — P. 2117.
- [3] Rahman, Muhammad Mahbubur. Unfolding the Structure of a Document using Deep Learning / Muhammad Mahbubur Rahman, Tim Finin // *arXiv preprint arXiv:1910.03678*. — 2019.
- [4] The representation of document structure: A generic object-process analysis / Dov Dori, David Doermann, Christian Shin et al. // Handbook of character recognition and document image analysis. — World Scientific, 1997. — Pp. 421–456.
- [5] Желамская, АА. Лингвистическая структура деловой документации на материале французского и итальянского языков / АА Желамская // *М.: Институт языкознания РАН*. — 2017.
- [6] Кушнерук, Сергей Петрович. Документная лингвистика / Сергей Петрович Кушнерук. — Общество с ограниченной ответственностью ФЛИНТА, 2011.
- [7] Заболеева-Зотова, АВ. Моделирование лексического анализа текста технического задания / АВ Заболеева-Зотова, ЮА Орлова // *Главный редактор сборника "Известия ВолгГТУ" д-р хим. наук, проф. член-корр. РАН ИА Новаков*. — 2007. — P. 40.
- [8] Орлова, Юлия Александровна. Автоматизация семантического анализа текста технического задания: Ph.D. thesis / Волгоградский государственный технический университет. — 2008.

- [9] *Juge, Rémi*. The fintoc-2019 shared task: Financial document structure extraction / Rémi Juge, Imane Bentabet, Sira Ferradans // Proceedings of the Second Financial Narrative Processing Workshop (FNP 2019). — 2019. — Pp. 51–57.
- [10] *Giguet, Emmanuel*. Daniel@ fintoc-2019 shared task: toc extraction and title detection / Emmanuel Giguet, Gaël Lejeune // Proceedings of the Second Financial Narrative Processing Workshop (FNP 2019). — 2019. — Pp. 63–68.
- [11] A Machine-Learning Based Approach for Extracting Logical Structure of a Styled Document. / Tae-young Kim, Suntae Kim, Sangchul Choi et al. // *TIIS*. — 2017. — Vol. 11, no. 2. — Pp. 1043–1056.
- [12] *Wexler, Michael C*. Structure extraction on electronic documents. — 2001. — 2. — US Patent 6,298,357.
- [13] Office Open XML File Formats — Fundamentals and Markup Language Reference. — 2016.
- [14] *Chen, Tianqi*. Xgboost: A scalable tree boosting system / Tianqi Chen, Carlos Guestrin // Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining. — 2016. — Pp. 785–794.
- [15] *Kuhner, Mary K*. Practical performance of tree comparison metrics / Mary K Kuhner, Jon Yamato // *Systematic biology*. — 2015. — Vol. 64, no. 2. — Pp. 205–214.
- [16] *Robinson, David F*. Comparison of phylogenetic trees / David F Robinson, Leslie R Foulds // *Mathematical biosciences*. — 1981. — Vol. 53, no. 1-2. — Pp. 131–147.
- [17] *Pal, Mahesh*. Random forest classifier for remote sensing classification / Mahesh Pal // *International journal of remote sensing*. — 2005. — Vol. 26, no. 1. — Pp. 217–222.
- [18] *Namboodiri, Anoop M*. Document structure and layout analysis / Anoop M Namboodiri, Anil K Jain // Digital Document Processing. — Springer, 2007. — Pp. 29–48.
- [19] *Paaß, Gerhard*. Machine learning for document structure recognition / Gerhard Paaß, Iuliu Konya // Modeling, Learning, and Processing of Text Technological Data Structures. — Springer, 2011. — Pp. 221–247.

- [20] *Pembe, F Canan*. A Tree Learning Approach to Web Document Sectional Hierarchy Extraction. / F Canan Pembe, Tunga Güngör // ICAART (1). — 2010. — Pp. 447–450.

Приложение 1

Манифест по разметке технических заданий (ТЗ)

Необходимо извлечь простую структуру из ТЗ, а именно:

- Титульный лист (title);
- Оглавление (toc);
- Структурные элементы (part);
- Элементы списка (item).

Задача представляет из себя классификацию изображений. Необходимо проанализировать набор изображений документа, на каждом изображении один параграф обведен в рамку.

Краткое описание типов параграфов:

- Все параграфы титульного листа — это **title**;
- Все параграфы оглавления — это **toc**;
- Все параграфы с номером/значком элемента списка — это **item**;
- Раздел/подраздел и выделенные элементы списка — это **part**;
- Обычный текст, не подошедший под описания выше — это **raw_text**;
- Нетекстовый элемент — это **other**.

Пример задания: Параграф, обведённый в красную рамку на рисунке 12, необходимо классифицировать.

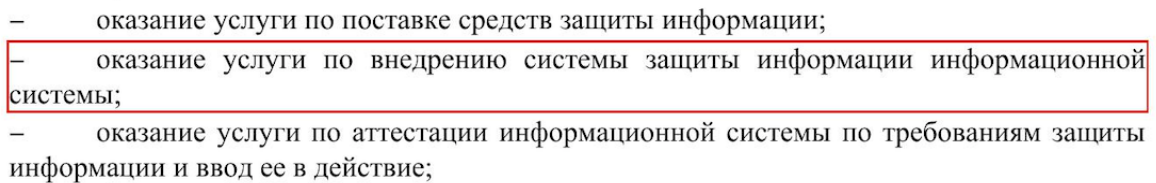
- 
- оказание услуги по поставке средств защиты информации;
 - оказание услуги по внедрению системы защиты информации информационной системы;
 - оказание услуги по аттестации информационной системы по требованиям защиты информации и ввод ее в действие;

Рис. 12: Пример задания

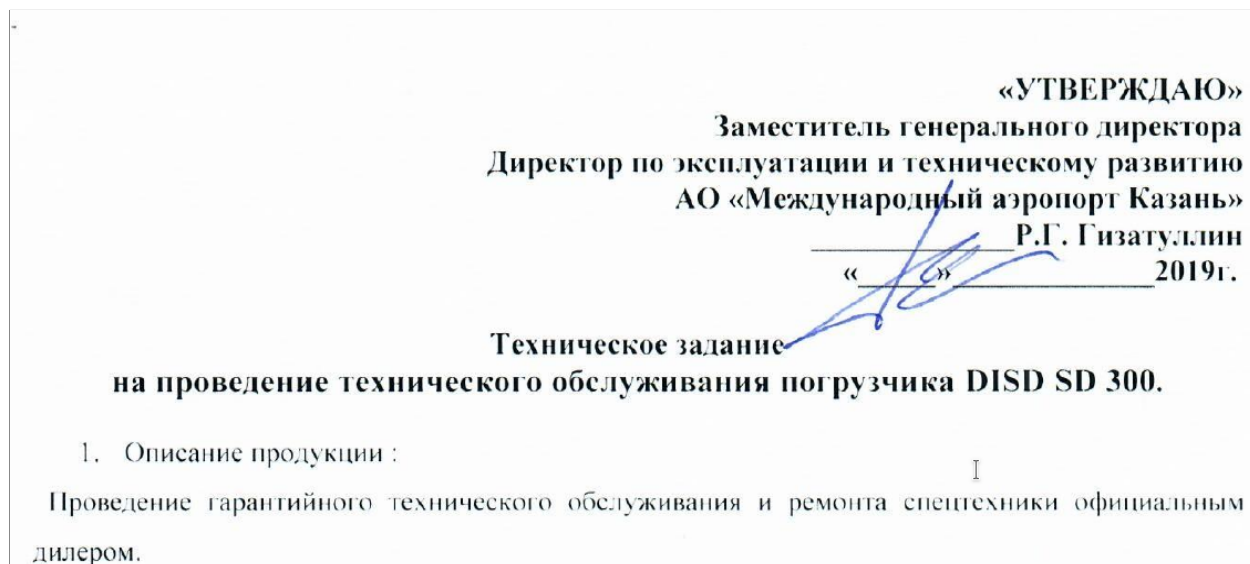


Рис. 14: Пример ТЗ с «шапкой»

ми списка являются параграфы, выделенные красной рамкой (остальные невыделенные строки — это `raw_text`).

Просто текст – это обычный текст, то есть всё остальное. На рисунке 18 `raw_text` выделен в красную рамку.

Other – если параграф вообще не является текстом.

Таким образом:

- Все параграфы титульника – это **title**.
- Все параграфы оглавления – это **toc**.
- Жирный выделяющийся текст, а особенно раздел и подраздел – это **part**.
- Все параграфы с номером/значком элемента списка – это **item**.
- Обычный текст, не попавший в пункты выше – это **raw_text**.
- Вообще не текст, а кусок рамки это **other**.

Содержание	
1	Общие сведения.....3
1.1	Основания для разработки.....3
1.2	Плановые сроки начала и окончания работ.....3
1.3	Сведения об использованных при создании СЗПДн АСУ Персоналом нормативно-технических документов.....3
1.4	Порядок оформления и предъявления заказчику результатов работ.....5
2	Назначение и цели создания СЗПДн АСУ Персоналом.....6
3	Характеристика объекта автоматизации.....7
4	Требования к СЗПДн АСУ Персоналом.....8
4.1	Требования к СЗПДн АСУ Персоналом в целом.....8
4.2	Требования к функциональным подсистемам СЗПДн.....15
4.3	Требования к видам обеспечения СЗПДн АСУ Персоналом.....18
5	Состав и содержание работ по созданию СЗПДн АСУ Персоналом....30
6	Порядок контроля и приемки СЗПДн АСУ Персоналом.....32
7	Требования к составу и содержанию работ по подготовке к вводу СЗПДн АСУ Персоналом в действие.....33
8	Требования к документированию.....34
	Список принятых сокращений.....35

Рис. 15: Пример оглавления

– оказание услуги по сопровождению системы защиты персональных данных.
<p>1.1. Требование по соответствию оказываемых услуг действующему законодательству Российской Федерации</p> <p>Оказываемые услуги и разрабатываемые документы должны соответствовать требованиям действующего на дату сдачи оказанных услуг законодательства Российской Федерации, в том числе нормативным и иным документам, приведенным в Приложении 1 к настоящему Техническому заданию.</p>
<p>9. Общие требования к лицензионному системному и прикладному программному обеспечению, средствам защиты информации, их качеству, техническим характеристикам и безопасности:</p> <p>9.1. Требования к средству защиты информации от несанкционированного Secret Net 7 (сетевой режим работ).</p> <p>Должно осуществлять:</p> <ul style="list-style-type: none"> защиту серверов и рабочих станций от НСД;

Рис. 16: Примеры части

<p>Проектная документация на информационную систему и (или) ее систему защиты информации подлежат согласованию с оператором информационной системы в случае, если он определен таковым в соответствии с законодательством Российской Федерации к моменту окончания проектирования системы защиты информации информационной системы персональных данных.</p> <p>При макетировании и тестировании системы защиты информации информационной системы в том числе осуществляются:</p> <ul style="list-style-type: none"> – проверка работоспособности и совместимости выбранных средств защиты информации с информационными технологиями и техническими средствами;
<p>9.2. Требования к программно-аппаратному комплексу "Соболь". Версия 3.0. или эквивалент.</p> <p>Должен осуществлять:</p>

Рис. 17: Примеры элементов списка

моделирования информационных систем и технологий виртуализации.

В случае если это предусмотрено действующим законодательством Российской Федерации, исполнитель должен иметь действующее на момент оказания услуг соответствующие специальные разрешения (лицензии) на такой вид деятельности. Такие специальные разрешения (лицензии) должны быть предоставлены до начала оказания соответствующей услуги.

1.4. Услуги по поставке средств защиты информации

Рис. 18: Пример обычного текста

Приложение 2

Манифест по сравнению пар строк

Изначально дано изображение документа, на котором два параграфа выделены в рамки. Необходимо отметить один из классов "больше"(greater), "меньше"(less), "равно"(equal).

Документы имеют определённую структуру, то есть одни параграфы документа вложены в другие. Необходимо уметь это определять. Таким образом, необходимо сравнить пары параграфов документа:

- Параграфы равны (equal);
- Первый параграф меньше второго (less);
- Первый параграф больше второго (greater).

Задача представляет из себя классификацию изображений. Необходимо проанализировать набор изображений документа, на каждом изображении два параграфа обведены в рамки: первый параграф обведён в красную рамку, второй – в синюю.

Краткое описание типов параграфов

- Параграфы не отличаются оформлением и являются одинаковыми структурными единицами — они равны (**equal**);
- Первый параграф выделяется оформлением больше, чем второй параграф — первый параграф больше второго (**greater**);
- Первый параграф является структурным элементом, в который включается второй параграф — первый параграф больше второго (**greater**);
- Первый параграф выделяется оформлением меньше, чем второй параграф — первый параграф меньше второго (**less**);
- Первый параграф включается во второй параграф — первый параграф меньше второго (**less**).

Пример задания:

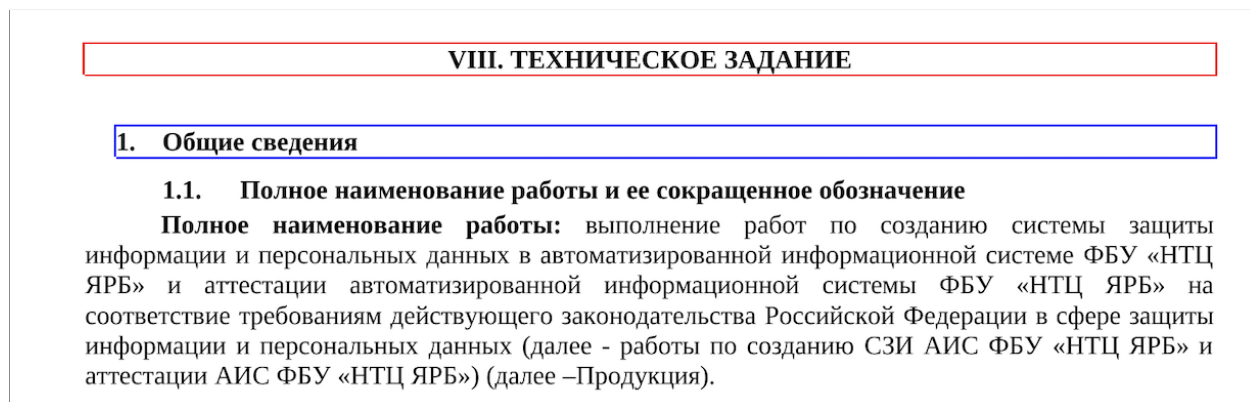


Рис. 19: Пример задания

Необходимо сравнить два параграфа, обведённые в красную и синюю рамки (рис. 19).

Равные параграфы (equal)

Параграфы могут быть равны, если выполнены следующие условия:

1. Визуально видно, что параграфы являются одинаковыми структурными единицами. Это могут быть списки с нумерацией одинакового типа или заголовки с одинаковым оформлением или простые текстовые строки.
2. Параграфы равны, если у них совпадает оформление. Это означает, что у них совпадает выравнивание (по центру, по левому краю и т. д.), размер шрифта, жирность, курсив, подчеркивание, отступ от левого края страницы.

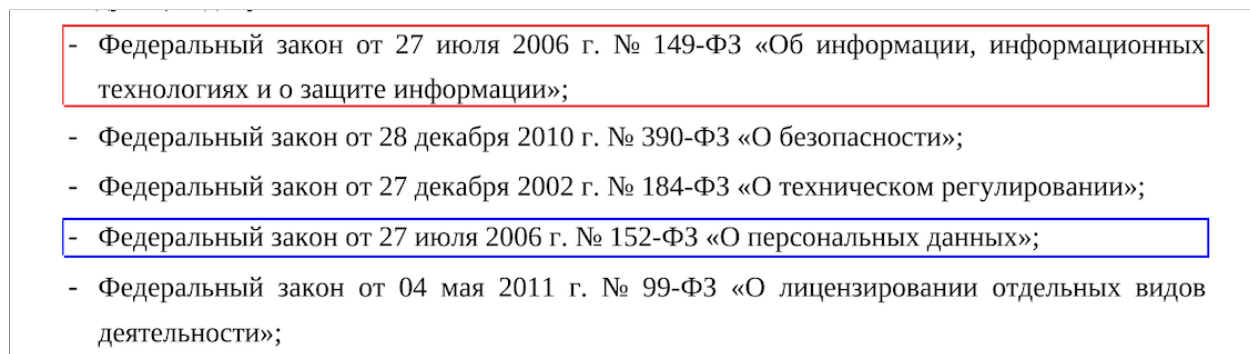


Рис. 20: Пример равных параграфов: элементы списка с одинаковым оформлением

В примере на рисунке 20 параграфы – это элементы списка с одинаковым оформлением.

1.5Срок выполнения работ	
Срок начала выполнения работ:	с даты заключения Договора.
Срок окончания выполнения работ:	120 календарных дней с даты заключения Договора.
<p>1.1. Требование по соответствию оказываемых услуг действующему законодательству Российской Федерации</p> <p>Оказываемые услуги и разрабатываемые документы должны соответствовать требованиям действующего на дату сдачи оказанных услуг законодательства Российской Федерации, в том числе нормативным и иным документам, приведенным в Приложении 1 к настоящему Техническому заданию.</p> <p>1.2. Услуги по формированию требований к защите информации, содержащейся в информационной системе</p> <p>Формирование требований к защите информации, содержащейся в информационной системе включает:</p> <ul style="list-style-type: none"> - классификацию информационной системы по требованиям защиты информации; 	

Рис. 21: Примеры равных параграфов: текстовые строки с одинаковым оформлением

В примерах на рисунке 21 параграфы – это текстовые строки с одинаковым оформлением.

Первый параграф больше второго (greater)

Первый параграф больше второго, если выполняется хотя бы одно из условий:

1. Первый параграф является структурной единицей, которая визуальнo и логически включает в себя второй параграф.
2. Первый параграф визуальнo «важнее» по оформлению, чем второй. Это означает, что выполнен один или несколько следующих пунктов (расположены по убыванию значимости):
 - Выравнивание первого параграфа по центру, у второго любое другое.
 - Шрифт первого параграфа больше шрифта второго.
 - Первый параграф жирный/курсивный/подчеркнутый, второй — нет, либо имеет меньше стилей.
 - Отступ от левого края первого параграфа меньше отступа второго.

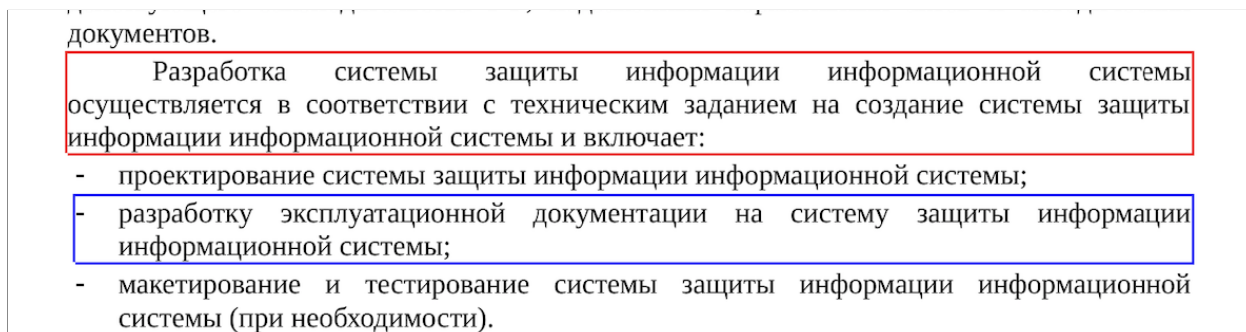


Рис. 22: Пример, в котором первый параграф больше второго (структурная вложенность элемента списка в параграф)

В примере на рис. 22 элемент списка (второй параграф, обведённый в синюю рамку) структурно вложен в параграф, обведённый в красную рамку.

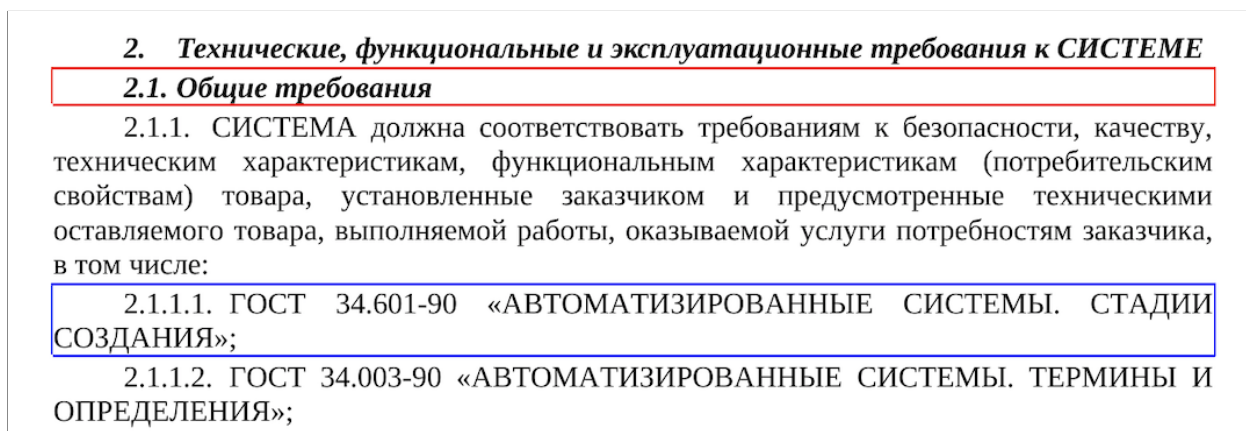


Рис. 23: Пример, в котором первый параграф больше второго (структурная вложенность элементов списка)

В примере на рис. 23 элемент списка в синей рамке структурно вложен в другой элемент списка в красной рамке (по нумерации).

В примерах на рисунке 24 первый параграф в красной рамке «важнее» по оформлению, чем второй в синей рамке.

Первый параграф меньше второго (less)

В данном случае всё аналогично предыдущему пункту, если первый и второй параграфы поменять местами.

В примере на рисунке 25 структурный элемент второго параграфа в синей рамке

<div style="border: 1px solid red; padding: 5px; text-align: center;"> ТЕХНИЧЕСКОЕ ЗАДАНИЕ </div> <div style="border: 1px solid blue; padding: 5px; text-align: center;"> на оказание услуг по разработке и внедрению автоматизированной системы «Адресная социальная помощь» </div> <p style="text-align: center;">ОГБУ «Многофункциональный центр предоставления государственных и муниципальных услуг Еврейской автономной области»</p>
<div style="border: 1px solid red; padding: 5px;"> Требования к объёму и характеристикам оказываемых услуг. </div> <div style="border: 1px solid blue; padding: 5px;"> 1. Объем оказываемых услуг: </div> <p>Состав и содержание услуг по созданию Системы защиты клиентского сегмента АИС МФЦ НО МФЦ на 18 окон (клиентских мест):</p>

Рис. 24: Примеры, в которых первый параграф больше второго (оформление)

<div style="border: 1px solid red; padding: 5px;"> 1.3. Плановые сроки начала и окончания работ Начало работ – дата подписания Контракта. Окончание работ – 31.07.2020 г. </div> <div style="border: 1px solid blue; padding: 5px;"> 1.4. Перечень НПА, на основании которых выполняются работы - Жилищный кодекс Российской Федерации от 29.12.2004 г. № 188-ФЗ; </div>

Рис. 25: Пример, в котором первый параграф меньше второго (структурная вложенность)

важнее первого в красной рамке.

В примере на рисунке 26 после вложенного списка в красной рамке начинается текст второго параграфа в синей рамке.

В примере на рисунке 27 по нумерации второй параграф в синей рамке является более важной структурной единицей, чем первый параграф в красной рамке.

Другое (other) Если один (или оба) параграф пустой или на изображении нет рамок (и т. д.), изображение следует проклассифицировать как other. Пример на рисунке 28.

Подытожим:

- разработку эксплуатационной документации на систему защиты информации информационной системы;
- макетирование и тестирование системы защиты информации информационной системы (при необходимости).

Система защиты информации информационной системы не должна препятствовать достижению целей создания информационной системы и ее функционированию.

При разработке системы защиты информации информационной системы учитывается ее информационное взаимодействие с иными информационными системами и информационно-телекоммуникационными сетями.

При проектировании системы защиты информации информационной системы:

Рис. 26: Пример, в котором первый параграф меньше второго (структурная вложенность)

2.1.1.6. ГОСТ Р ИСО/МЭК 15271 -02 «ПРОЦЕССЫ ЖИЗНЕННОГО ЦИКЛА ПРОГРАММНЫХ СРЕДСТВ»;

2.1.1.7. ГОСТ Р ИСО/МЭК 15910-2002 «ПРОЦЕСС СОЗДАНИЯ ДОКУМЕНТАЦИИ ПОЛЬЗОВАТЕЛЯ ПРОГРАММНОГО СРЕДСТВА»

2.2. Требования по диагностированию СИСТЕМЫ

2.2.1. Система должна предоставлять инструменты мониторинга основных параметров состояния и работоспособности подсистем, ведения журнала действий пользователей и прочих действий, влияющих или изменяющих состояние системы, ее компонентов или учетных параметров.

2.2.2. Должен быть предоставлен пользовательский интерфейс для возможности просмотра диагностических событий, мониторинга процесса функционирования системы.

Рис. 27: Пример, в котором первый параграф меньше второго (структурная вложенность элементов списка)

от _____ 2015 г.

Техническое задание на оказание услуг по созданию системы защиты персональных данных на автоматизированных рабочих местах комитета агропромышленного комплекса Курской области

Рис. 28: Пример, в котором один из параграфов пустой

- Параграфы не отличаются оформлением и структурой – это equal.
- Первый параграф является более важной структурной единицей или «важнее»

оформлен – это greater.

- Случай, обратный предыдущему – это less.