



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:**

Козлов Илья Сергеевич  
*Научный консультант:*  
Гомзин Андрей Геннадьевич

Москва, 2020

## **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.

## **Abstract**

Hierarchical logical structure extraction  
from text documents in docx format

*Bogatenkova Anastasia 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 structure extraction from such documents.

# Содержание

<b>1 Введение</b>	<b>5</b>
<b>2 Постановка задачи</b>	<b>6</b>
<b>3 Обзорная часть</b>	<b>7</b>
3.1 Обзор по представлениям логической структуры документов . . . . .	7
3.1.1 Представление структуры документа в виде дерева . . . . .	8
3.1.2 Представление структуры документа в виде графа . . . . .	8
3.1.3 Представление структуры документа с использованием формальных грамматик . . . . .	10
3.1.4 Представление структуры документа в виде зон и логических меток	10
3.2 Обзор по форматам документов . . . . .	11
3.2.1 SGML и основанные на нем форматы . . . . .	11
3.2.2 XML и основанные на нем форматы . . . . .	12
3.2.3 ODA (Open Document Architecture) . . . . .	13
3.2.4 JSON и YAML . . . . .	14
3.2.5 TeX . . . . .	14
3.2.6 PDF . . . . .	15
3.3 Существующие методы извлечения иерархической структуры из документов . . . . .	15
<b>4 Общий план по извлечению структуры</b>	<b>19</b>
<b>5 Особенности формата docx</b>	<b>20</b>
5.1 Существующие библиотеки по работе с форматом . . . . .	20
5.1.1 python-docx . . . . .	20
5.1.2 docx2python . . . . .	21
5.1.3 pydocx . . . . .	21
5.2 Составляющие docx-файла . . . . .	22
5.3 Некоторые понятия, используемые в документации формата . . . . .	23
5.3.1 Тело документа (body) . . . . .	23
5.3.2 Параграф (paragraph) . . . . .	24

5.3.3	Текстовый элемент (run) . . . . .	25
5.3.4	Стиль (style) . . . . .	26
5.3.5	Абстрактная и непосредственная нумерация (abstract numbering and numbering) . . . . .	27
<b>6</b>	<b>Извлечение метаданных и текста из docx документов</b>	<b>31</b>
6.1	Реализация обработчика docx документов . . . . .	31
6.2	Тестирование обработчика docx документов . . . . .	34
<b>7</b>	<b>Реализация метода извлечения структуры</b>	<b>35</b>
7.1	Описание извлекаемой структуры . . . . .	35
7.2	Создание обучающего набора документов . . . . .	37
7.2.1	Создание изображений для разметки из docx файла . . . . .	38
7.2.2	Манифесты для классификаторов . . . . .	40
7.2.3	Набор данных для разметки и процесс разметки . . . . .	40
7.3	Реализация метода определения типа параграфов документа . . . . .	42
7.3.1	Извлечение признаков . . . . .	42
7.3.2	Обучение классификатора . . . . .	43
7.3.3	Постобработка . . . . .	44
7.4	Реализация метода построения иерархической структуры документа . . . . .	45
7.4.1	Извлечение признаков . . . . .	46
7.4.2	Обучение классификатора . . . . .	47
7.4.3	Построение дерева документа . . . . .	49
<b>8</b>	<b>Заключение</b>	<b>51</b>
	<b>Список литературы</b>	<b>52</b>
	<b>Приложение 1</b>	<b>55</b>
	<b>Приложение 2</b>	<b>60</b>
	<b>Приложение 3</b>	<b>66</b>

# 1 Введение

Документы имеют определённую логическую структуру. Например, законы делятся на главы, статьи, разделы. Научные статьи состоят из аннотации, введения, обзора существующих работ и других секций. Информация о логической структуре полезна для автоматического анализа документа.

В настоящее время большое количество документов создается, редактируется и хранится в формате docx. Однако существующие библиотеки по работе с docx документами предназначены в первую очередь для создания и редактирования документов, а не для их 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.

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. Произвольный пользователь может создать структурированный документ без использования специальных возможностей, предоставляемых офисными приложениями. Поэтому логическая структура в docx документах далеко не всегда присутствует явным образом.

Таким образом, задача обработки документов в формате docx и последующего извлечения логической структуры актуальна и может быть полезна во многих сферах деятельности, связанных с работой с подобными документами.

## 2 Постановка задачи

Задача состоит в извлечении логической структуры из документов в docx формате. Существуют различные способы представления структуры документа. Мы будем извлекать иерархическую структуру в виде дерева, так как многие документы состоят из последовательности вложенных друг в друга частей. Для определения логической структуры следует выбрать домен, с которым предстоит работать. Мы будем извлекать логическую структуру из документов типа «техническое задание».

В рамках поставленной задачи необходимо выполнить следующее:

1. Провести обзор способов представления логической структуры документа;
2. Провести обзор некоторых форматов документов;
3. Описать существующие методы извлечения структуры из документов;
4. Описать особенности формата docx;
5. Реализовать извлечение текста и необходимых метаданных из документов в формате docx;
6. Описать извлекаемую логическую структуру;
7. Создать обучающий набор документов и осуществить его разметку;
8. Реализовать метод определения типа строк документа;
9. Реализовать метод построения иерархической структуры на основе сравнения пар строк.

### **3 Обзорная часть**

В подсекции 3.1 дадим определения основных типов структуры документа и опишем различные точки зрения на то, в каком виде можно представлять логическую структуру документа. В подсекции 3.2 рассмотрим некоторые из форматов, которые использовались и используются для хранения и представления логической и физической структуры документа. Сравнительную таблицу по форматам можно посмотреть в приложении. И, наконец, в подсекции 3.3 рассмотрим существующие методы извлечения логической структуры из документов.

#### **3.1 Обзор по представлениям логической структуры документов**

Существует огромное количество документов самых разных форматов, доменов и имеющих различное устройство. Однако для всех документов можно определить три основных типа структуры, которая из них выделяется: физическая, логическая и семантическая [1].

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 подразумевает некоторый анализ физических составляющих документа. Данный тип структуры связан с конкретным доменом, к которому документ относится, так как для разных доменов физические части документа выполняют разные функции и могут иметь разный семантический смысл (так, в научных статьях можно выделить введение, обзор существующих работ, заключение, список литературы). Выделение логической структуры документа

подразумевает определение этих функций и семантического смысла конкретных составляющих документа, объединение в соответствии с этим физических частей документа (например, символы объединяются в слова и строки, строки объединяются в текстовые блоки). Кроме того, определяется взаимодействие частей друг с другом в физическом смысле, например, определяется порядок чтения или вложенность одного текстового блока в другой (многоуровневые списки или заголовки разных уровней). Логическая структура не анализирует смысл предложений, их взаимодействие друг с другом в семантическом смысле (например, в одном из предложений приводится пример, который описывается более подробно в следующем предложении). 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. Задачами выявления семантики текста занимается отдельная область – обработка естественного языка.

В рамках решаемой задачи рассматривается логическая структура документа. Опишем некоторые из способов, которые применяются для представления такой структуры.

### **3.1.1 Представление структуры документа в виде дерева**

Большое количество документов представляет из себя последовательность вложенных друг в друга частей. Например, статьи состоят из секций, которые, в свою очередь, могут состоять из подсекций и т. д., законы могут состоять из глав, статей, пунктов и т. д. Дерево помогает получить представление документа в виде иерархической структуры, то есть документ разбивается на последовательность вложенных друг в друга элементов [2, 3] (рис. 1). На каждом уровне иерархии находятся элементы определенных типов. В листовых вершинах, как правило, располагается простой текстовый блок.

### **3.1.2 Представление структуры документа в виде графа**

Произвольный граф позволяет представить разбиение документа на части (каждая часть является вершиной графа), а также описать порядок чтения частей [4] (ребра

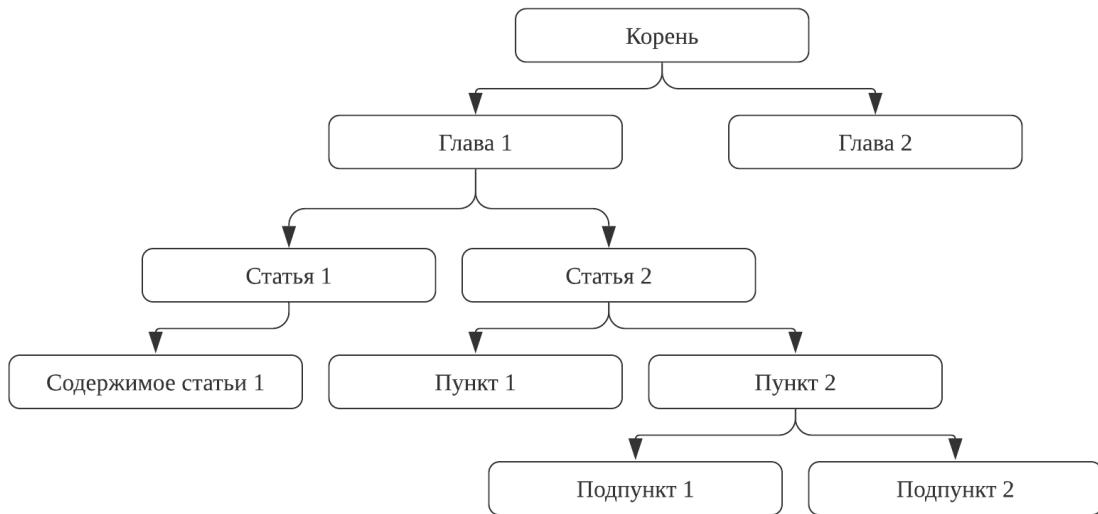


Рис. 1: Пример структуры документа в виде дерева

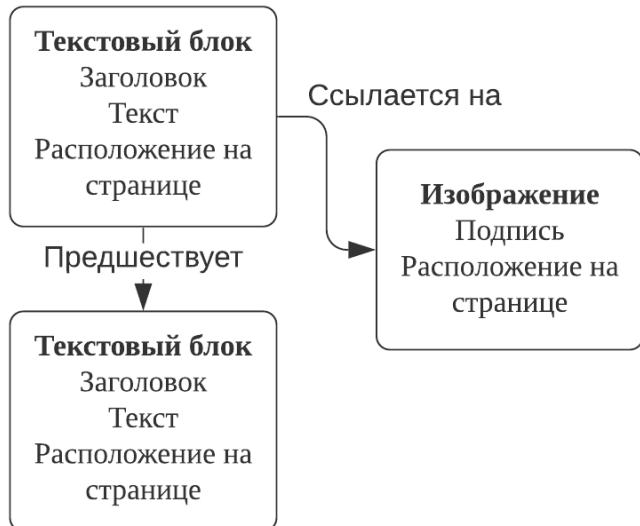


Рис. 2: Пример структуры документа в виде графа

графа могут быть помечены и описывать тип взаимоотношений между частями документа). Структура документа при этом может получиться не обязательно иерархической (рис. 2). Так, документ можно разбить на текстовые блоки, изображения, таблицы и определить порядок чтения этих элементов в документе.

### 3.1.3 Представление структуры документа с использованием формальных грамматик

```
[0.5] < START > → < TITLE > < COLUMN > < COLUMN >
[0.5] < START > → < TITLE > < COLUMN >
[1.0] < TITLE > → < text_line >
[1.0] < COLUMN > → < TEXT_BLOCKS >
[0.8] < TEXT_BLOCKS > → < TEXT_BLOCK > < space > < TEXT_BLOCKS >
[0.2] < TEXT_BLOCKS > → < TEXT_BLOCK >
[1.0] < TEXT_BLOCK > → < TEXT_LINES >
[0.9] < TEXT_LINES > → < text_line > < newline > < TEXT_LINES >
[0.1] < TEXT_LINES > → < text_line >
```

(a)

```
< START > → < TITLE > < COLUMN >
→ < text_line > < COLUMN >
→ < text_line > < TEXT_BLOCKS >
→ < text_line > < TEXT_BLOCK > < space > < TEXT_BLOCKS >
→ < text_line > < text_line > < newline > < TEXT_BLOCK > < space > < TEXT_BLOCKS >
→ < text_line > < text_line > < newline > < text_line > < space > < TEXT_BLOCKS >
→ < text_line > < text_line > < newline > < text_line > < space > < TEXT_BLOCK >
→ < text_line > < text_line > < newline > < text_line > < space > < text_line >
```

(b)

Рис. 3: Пример структуры документа в виде формальной грамматики: (а) пример стохастической контекстно-свободной грамматики, которая выводит текстовый документ с заголовком. Заглавными буквами помечены нетерминальные символы, строчными буквами обозначены терминальные символы. (б) пример вывода документа, состоящего из заголовка и трех строк, из грамматики, представленной в (а).

Документ может быть представлен последовательностью правил, которые необходимо обработать с помощью специального парсера [5]. В результате такой обработки получается исходный документ (рис. 3).

### 3.1.4 Представление структуры документа в виде зон и логических меток

Документ может быть представлен как плоская структура: последовательность частей какого-либо типа [6]. Такими частями могут быть страницы, текстовые блоки, строки, слова, символы и т. д. В зависимости от требований того или иного домена, каждому выделенному элементу назначается семантическая метка. Так, документ можно рассматривать построчно и каждой строке присваивать определенный тип, например,

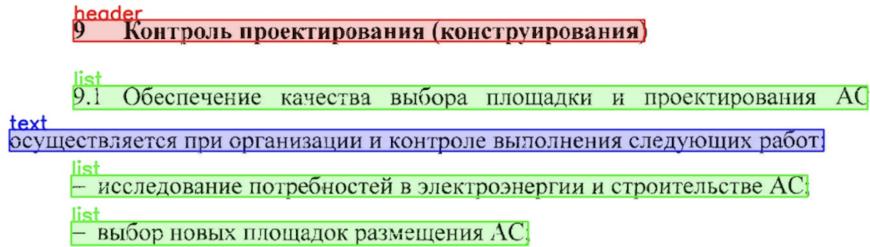


Рис. 4: Пример структуры документа в виде списка помеченных строк

«заголовок», «элемент списка» и «простая текстовая строка» (рис. 4) [7].

Помимо плоской структуры, семантические метки могут применяться и в более сложных структурах. Например, в иерархической структуре в виде дерева узлы могут быть типизированы и иметь некоторый семантический смысл.

## 3.2 Обзор по форматам документов

### 3.2.1 SGML и основанные на нем форматы

- SGML [8] – язык, использующий разметку (дополнительные аннотации в содержимом документа). Стандартизован ISO в 1986 году.

SGML документ состоит из трех файлов:

1. DTD (определение типа документа),
2. SGML декларация (описание символов, используемых в DTD и тексте документа),
3. экземпляр документа (текст документа + ссылка на DTD).

SGML предназначался для описания только логической структуры документа.

- DAFS [1] предназначен для представления изображений документов и результатов распознавания таких документов. DAFS за основу берет SGML, однако расширяет его, позволяя хранить не только логическую структуру. Документ представляет собой иерархию вложенных друг в друга сущностей (документ, глава, блок), при этом есть два дерева - для логической и физической структуры, листья деревьев с содержимым документа общие. Сущность может иметь несколько предков, поэтому структура документа может иметь несколько вариантов иерархии. Это может

использоваться для представления логической и физической структуры, а также могут храниться альтернативные структуры (если структура извлекалась не со 100% точностью).

### 3.2.2 XML и основанные на нем форматы

- XML [9] – расширяемый язык разметки, являющийся подмножеством SGML. Расширяемый, так как не фиксируется конкретная разметка. XML-процессор (парсер) — программа, анализирующая разметку и передающая информацию о структуре документа другой программе — приложению. XML унаследовал от SGML описание типов с помощью DTD (но это не обязательно). Существуют парсеры, проверяющие соответствие документа типу, описанному в DTD.

Основное различие между SGML и XML версиями состоит в следующем:

- XML элементы должны быть всегда закрыты.
- XML элементы должны иметь правильную вложенность.
- значения атрибутов должны быть обязательно в кавычках.
- DocBook [10] – стандарт, разработанный в основном для технической документации. Не предназначен для описания визуального представления документа, этим занимаются специальные утилиты. DocBook может быть основан на SGML или на XML.
- DITA [11] – специфический стандарт для работы с содержимым документов.
  - Основной единицей содержимого в DITA является топик – «озаглавленный блок информации, который может быть понят в отдельности от других блоков и используется в различных контекстах».
  - DITA-карта представляет собой список указателей на набор конкретных топиков, определяющий, какие именно топики должны быть включены в документ. Также карта определяет порядок и иерархию топиков и обеспечивает навигацию.
- HTML, XHTML, HTML5 [12] – языки для представления содержимого всемирной паутины. HTML — теговый язык разметки документов. Набор тегов и атрибутов

указан в DTD-описании типа документа, общем для всех документов для конкретной версии HTML. Язык HTML до пятой версии соответствовал SGML стандарту, XHTML соответствует стандарту XML (более строгие требования к правильности документов), HTML5 является расширением SGML, от предыдущих версий отличается набором элементов и атрибутов, синтаксисом и поддержкой специальных данных (математических формул векторной графики). HTML5 был создан как единый язык разметки, который мог бы сочетать синтаксические нормы HTML и XHTML.

- ALTO [13] – спецификация XML, предназначенная для хранения документов, обработанных с помощью систем оптического распознавания символов. Зачастую используется вместе с METS (является его частью). METS - спецификация XML, предназначенная для описания метаданных документа.
- TEI [13] – формат, предназначенный для кодирования литературных и лингвистических текстов. Делается упор на семантику и логическую структуру документов, а не на визуальное представление. Основа документа - рекурсивно вложенные друг в друга элементы div, семантический смысл которых указан в атрибутах. TEI используется во многих сферах, поэтому спецификация очень обширна. Однако предоставляется возможность приспособить стандарт для конкретной предметной области.
- Office Open XML (OOXML, DOCX, XLSX, PPTX) [14] – серия форматов файлов для хранения электронных документов пакетов офисных приложений — в частности, Microsoft Office. Формат представляет собой zip-архив, содержащий текст в виде XML, графику и другие данные. Форматы предназначены прежде всего для отображения физической структуры документа. Более подробная информация о docx находится в подсекции 5.2.

### 3.2.3 ODA (Open Document Architecture)

Язык описания структурированных данных объектного типа [15]. Язык более сложный, чем SGML, так как помимо логической структуры он описывает также физическую структуру.

Основные концепции ODA:

1. Логическая и физическая структуры документа (содержимое из логической структуры связывается с объектами физической структуры);
2. Специфическая (конкретная реализация) и общая (шаблонная) структуры документа;
3. Класс документа - набор характеристик, присущих отпределенной категории документов.

### 3.2.4 JSON и YAML

JSON (JavaScript Object Notation) [16] - простой формат обмена данными. Он основан на подмножестве языка программирования JavaScript. JSON основан на двух структурах данных:

- Коллекция пар ключ/значение;
- Упорядоченный список значений.

YAML [17] - формат, удобный для сериализации данных для всех языков программирования. YAML можно рассматривать как "расширенный JSON". Существуют некоторые отличия между данными форматами, например, возможность комментирования внутри файла, поддержка расширяемых типов данных, поддержка блочного синтаксиса с отступами, механизм для создания переменных, на которые можно ссылаться.

### 3.2.5 TeX

TeX [18] - Система компьютерной вёрстки для создания компьютерной типографии. В основном используется в научных документах, однако может подойти практически для любого домена. Основной замысел системы верстки заключается в том, что пользователь имеет возможность сосредоточиться на содержимом документа, практически не отвлекаясь на детали визуального представления. Автор документа может описывать различные логические структуры: глава, секция, таблица, рисунок, список и т. д., в тоже время имеется возможность описания физической структуры документа вручную.

### **3.2.6 PDF**

PDF (Portable Document Format) [19] - это формат файла, созданный Adobe Systems для обмена документами. PDF используется для представления страниц документов способом, независимым от прикладного программного обеспечения, оборудования и операционной системы. В настоящее время формат PDF широко используется для длительного хранения и архивирования документов в электронной форме. PDF можно генерировать в любом ПО для обработки документов, чтобы получить точное и фиксированное визуальное представление исходного документа. Однако PDF ориентирован на сохранение внешнего вида документа и не гарантирует сохранения его физической и логической структуры.

## **3.3 Существующие методы извлечения иерархической структуры из документов**

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 вычленить 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.

В статье [20] 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 TET, 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 статье [6] 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, повестки) 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.

В патенте [2] 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 (заголовок, список, обычный параграф), которые удается 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 (определенный 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.

## 4 Общий план по извлечению структуры

Формат docx позволяет описать физическую структуру документа, на основе которой необходимо построить логическую структуру. Документы в формате docx состоят из структурных единиц, называемых параграфами. Поэтому основным элементом выделяемой логической структуры логично сделать параграф. В качестве выделяемой структуры выберем дерево с типизированными узлами. Ограничим область обрабатываемых документов техническими заданиями и далее выделим основные типы параграфов в технических заданиях и принцип построения их иерархии.

Таким образом, поставим задачу как классификацию параграфов документа на несколько типов и построение дерева типизированных параграфов. Для решения поставленной задачи предлагается следующий план действий:

1. Извлечение необходимой информации из документов: текст и метаданные, такие как размер шрифта, отступ от края страницы, выравнивание, жирность шрифта, курсив и подчеркивание.
2. Формирование вектора признаков на основе извлеченных метаданных для задачи классификации параграфов документа.
3. Обучение классификатора параграфов документа.
4. Формирование вектора признаков на основе извлеченных метаданных для задачи сравнения пар параграфов документа.
5. Обучение компаратора пар параграфов документа.
6. Построение дерева типизированных параграфов документа на основе классификатора и компаратора параграфов.

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. Главной причиной выбора данного языка программирования является наличие библиотек с реализацией алгоритмов машинного обучения, таких как sklearn [21] и xgboost [22].

## 5 Особенности формата docx

### 5.1 Существующие библиотеки по работе с форматом

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-docx [23] – 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.
- 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

Docx2python [24] – 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.

### 5.1.3 pydocx

Pydocx [25] – 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.

Подводя итог, стоит заметить, что каждая из библиотек позволяет извлечь из docx документов некоторую информацию, которая частично удовлетворяет требованиям нашей задачи. 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.

Таким образом, следует изучить устройство docx формата и реализовать обработку документов и извлечение всей описанной выше информации.

## 5.2 Составляющие docx-файла

Документация по docx формату содержится в специально разработанном стандарте [14]. Docx-файл — это zip-архив который физически содержит 2 типа файлов:

- xml файлы с расширениями xml и rels;
- 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 обрамляется 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 `ilvl` 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 шеснадцатеричный 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. 5. 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.

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



Рис. 5: Порядок применение настроек свойств

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 ilvl), 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 выделяются 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 реали-

зациях 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 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 OpenOffice.org.

ных 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 lvl 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. Данний 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;
- rPr – 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 сбросывание нумерации того абстрактного списка, наследником которого является список, содержащий этот тег. Значение атрибута val показывает новое начальное значение нумерации.

## 6 Извлечение метаданных и текста из docx документов

### 6.1 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 главе 5 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 методанными.

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.

## 6.2 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 закупок [26]. Это позволило увеличить разнообразие документов для проверки безошибочной работы обработчика.

## 7 Реализация метода извлечения структуры

### 7.1 Описание извлекаемой структуры

Существует несколько видов структуры, которую можно извлекать из документов, некоторые из которых описаны в секции 3.1. Мы будем извлекать иерархическую структуру в виде дерева (подсекция 3.1.3). При этом, каждому узлу дерева будет приписана метка, описание меток будет дано ниже.

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.

<b>Название документа</b> (0 уровень)
<b>Глава 1</b> (1 уровень)
Содержимое главы 1 (2 уровень)
<b>Статья 1</b> (2 уровень)
Содержимое статьи 1 (3 уровень)
• Первый элемент маркированного списка (4 уровень)
• Второй элемент маркированного списка (4 уровень)
<b>Статья 2</b> (2 уровень)
Содержимое статьи 2 (3 уровень)
<b>Глава 2</b> (1 уровень)
1. Первый элемент нумерованного списка (2 уровень)
2. Второй элемент нумерованного списка (2 уровень)

Рис. 6: Пример документа для извлечения структуры

На рисунке 6 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 6. 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:

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 6, будет построено дерево, изображенное на рисунке 7.



Рис. 7: Пример построенного дерева на основе документа на рисунке 6

## 7.2 Создание обучающего набора документов

В главе 4 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. Если нет готового набора документов для обучения, нужно его создать, разметив документы вручную.

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, необходимо преобразовать docx документ в набор изображений с выделенными параграфами и установить соответствие между изображениями и параграфами.

графами документа.

### 7.2.1 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, обведенными в рамку (см. рисунок 8).

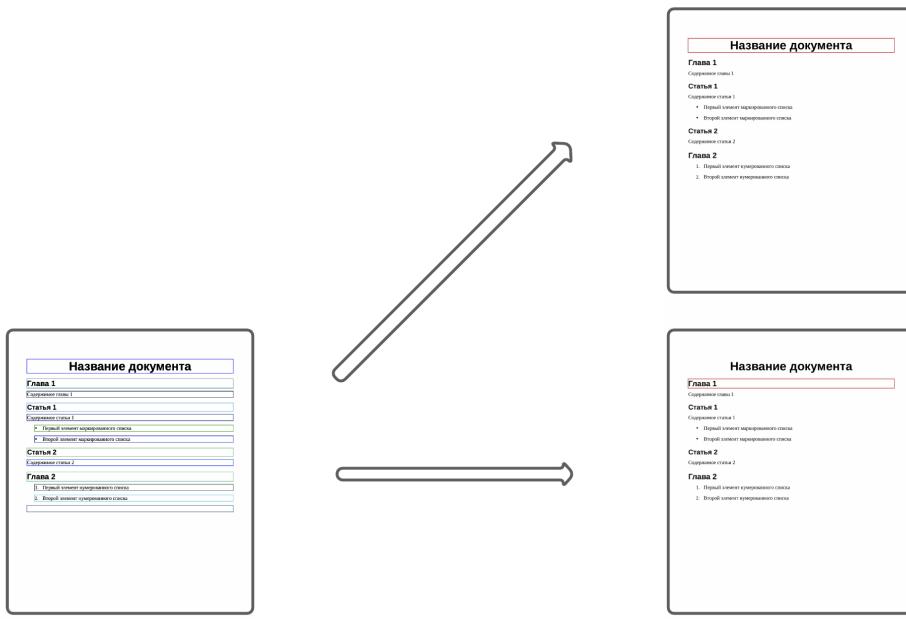
В общих чертах алгоритм состоит из следующих шагов:

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).

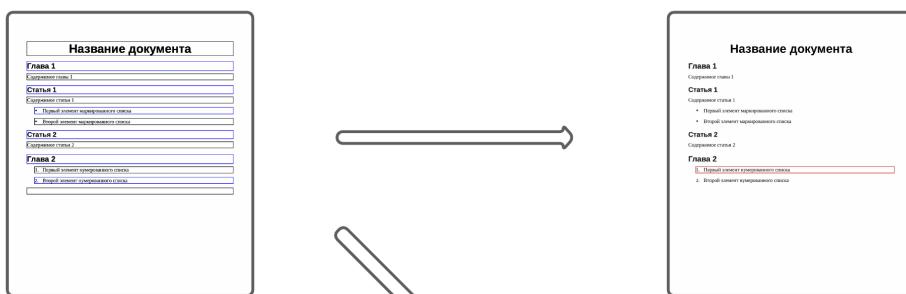
В docx документах параграфы обводились в рамку с помощью вставок специальных тегов в файл document.xml. Изменённые таким образом документы конвертируются в pdf документы. При создании документов с рамками, цвета рамок запоминаются.

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.



Документ с рамками разных цветов

Документы с одним параграфом,  
обведённым в рамку



Документ с рамками двух цветов

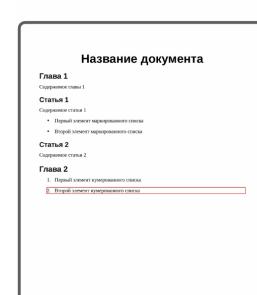


Рис. 8: Создание изображений для разметки из docx документа

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.

### 7.2.2 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.

Для классификации параграфов ТЗ документов необходимо определить типы параграфов для выделения и описать на примерах, какие параграфы относятся к тому или иному типу. В приложении 2 описаны указания по разметке документов и 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 3.

### 7.2.3 Набор данных для разметки и процесс разметки

Набор данных представляет собой набор из 22 документов, скачанных с сайта государственных закупок [26]. 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 РАН [27].

Получившееся распределение параграфов по типам представлено на рисунке 9.

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

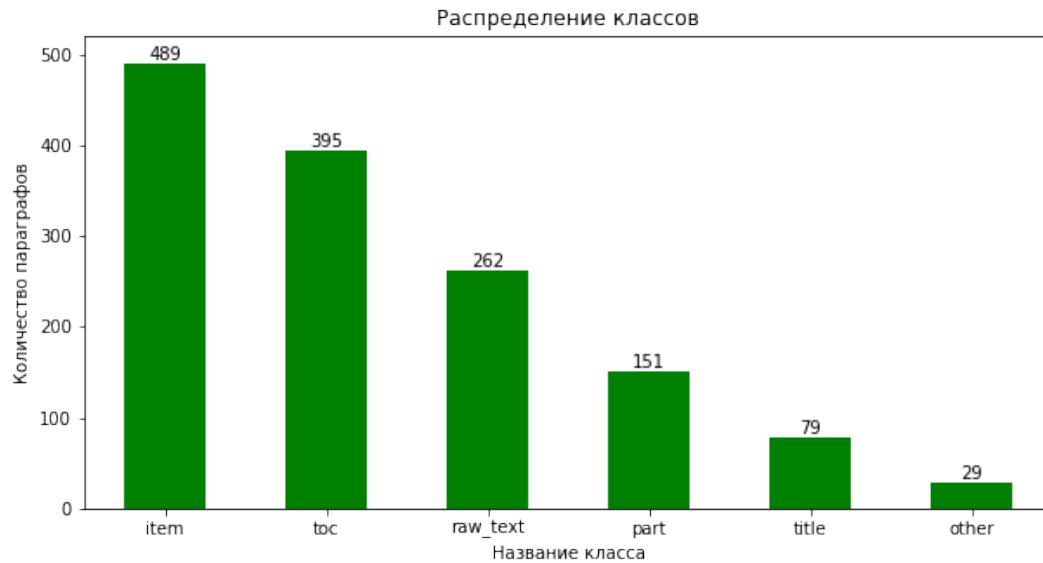


Рис. 9: Распределение параграфов по типам

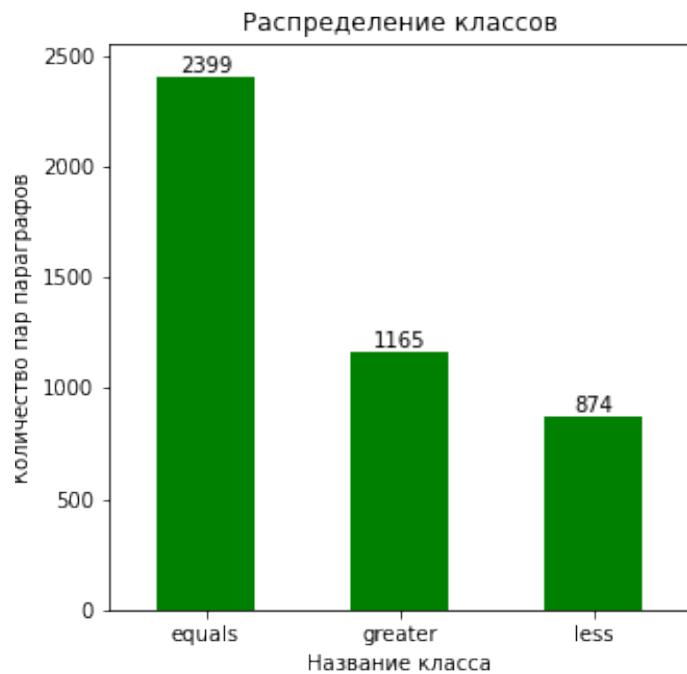


Рис. 10: Распределение типов пар параграфов

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 10.

## 7.3 Реализация метода определения типа параграфов документа

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.

### 7.3.1 Извлечение признаков

С помощью реализованного обработчика docx документов (см. главу 6) 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).

### 7.3.2 Обучение классификатора

В качестве классификатора использовался XGBoost [22], являющийся одним из самых эффективных алгоритмов машинного обучения на данный момент. Нейронные сети для решения поставленной задачи не походят, 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.

### 7.3.3 Постобработка

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

В процессе кросс-валидации было выявлено разбиение на тренировочное и валидационное множества, при котором точность предсказания была минимальной. Для такого разбиения была построена матрица ошибок (рисунок 11).

Исходя из матрицы ошибок можно заключить, что чаще всего путаются item и part

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

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

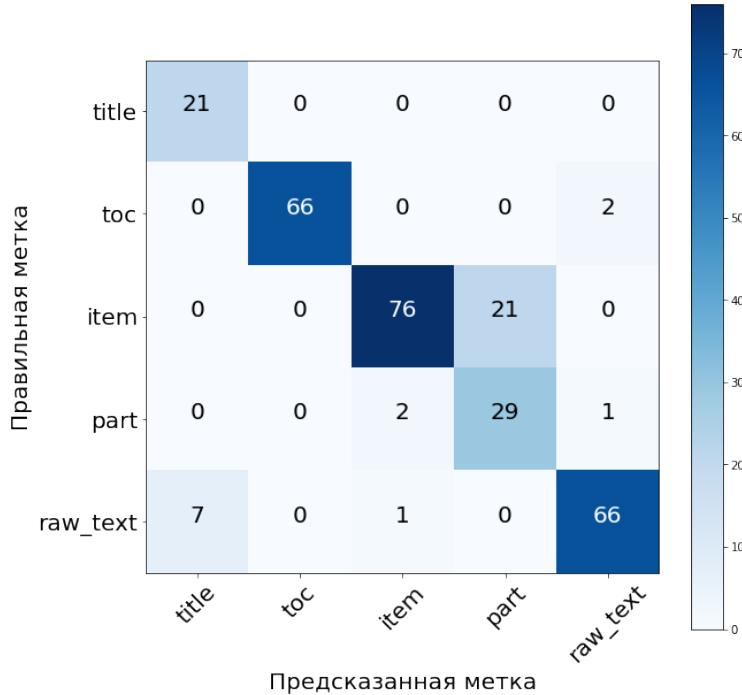


Рис. 11: Матрица ошибок классификатора типов параграфов

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.

## 7.4 Реализация метода построения иерархической структуры документа

Общая схема метода построения иерархической структуры документа выглядит следующим образом:

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

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.

#### 7.4.1 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.

#### 7.4.2 Обучение классификатора

Аналогично случаю классификатора типов параграфов для классификации пар строк использовался XGBoost [22].

В результате подбора параметров алгоритма и корректировки признаков после кросс-валидации были получены результаты, описанные в таблице 3. Разбиение документов на тренировочное и валидационное множества производилось 10 раз, в таблице представлены усреднённые значения точности и F1-меры.

Accuracy	F1-measure (macro)
0.9809	0.97848

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

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

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 визуально незаметно выравнивание по центру, которое играет большую роль при классификации (из-за выравнивания по центру первого параграфа классификатор считает, что он больше второго). Еще 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».

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

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.

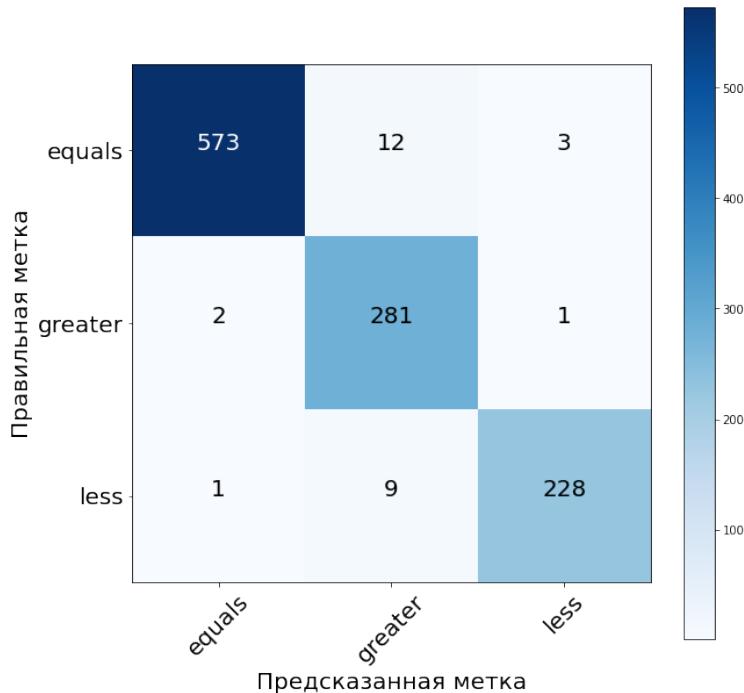


Рис. 12: Матрица ошибок классификатора пар параграфов

В процессе кросс-валидации было выявлено разбиение на тренировочное и валида-

ционное множества, при котором F1-мера предсказания была минимальной. Для такого разбиения была построена матрица ошибок (рисунок 12). 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.

#### 7.4.3 Построение дерева документа

На основе реализованного компаратора строк написан код для построения дерева документа в соответствии с алгоритмом из подсекции 7.1. Структура документа извлекается в виде словаря, который легко перевести в json формат для сохранения данных (см. листинг 1).

Листинг 1: Структура узла извлекаемого дерева

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

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 [28]. Пример визуализации дерева документа без типов показан на рис. 13.

Для документов типа «Техническое задание» реализовано отдельное построение дерева, в котором все параграфы, относящиеся к заголовку, становятся вложенными в

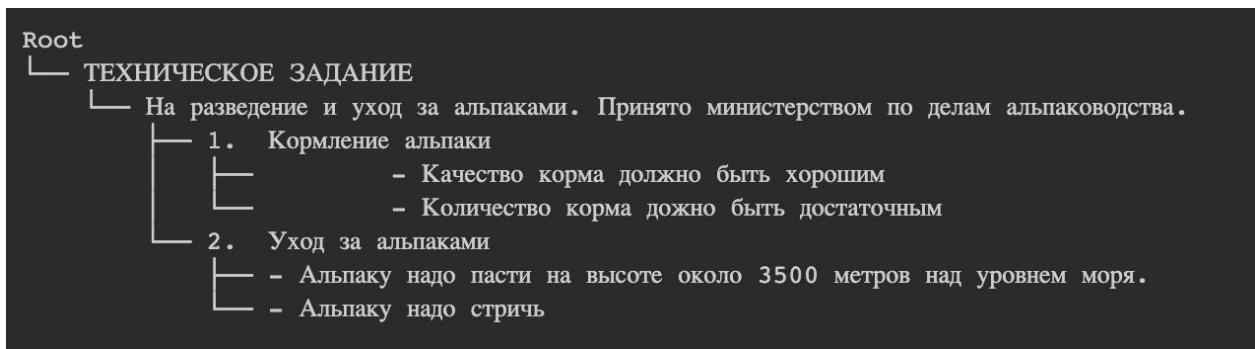


Рис. 13: Пример визуализации дерева документа

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. 14.

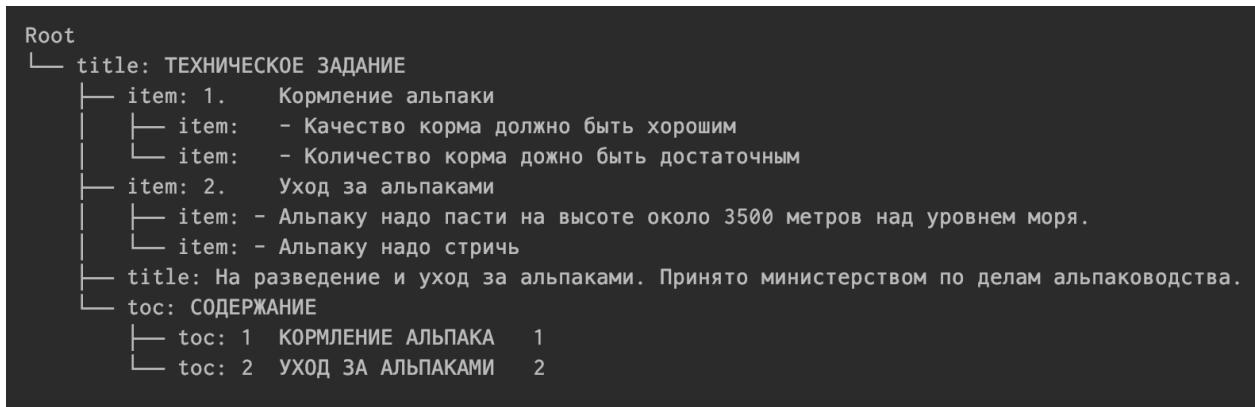


Рис. 14: Пример визуализации типизированного дерева для технического задания

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.

## 8 Заключение

Таким образом, в рамках поставленной задачи были проведены следующие обзоры: обзор способов представления логической структуры документа и обзор некоторых форматов документов для структурирования информации и составлена сравнительная таблица форматов. Описаны некоторые существующие методы извлечения логической структуры из документов и особенности формата docx. Реализован обработчик docx документов, позволяющий получать из docx документов текст с дополнительными метаданными.

Описана конкретная извлекаемая логическая структура и составлен обучающий набор данных для извлечения структуры. Реализованы метод определения типа строк технических заданий и метод построения иерархической структуры в виде дерева на основе компаратора строк. Результат построения дерева можно визуализировать для лучшего понимания структуры документа.

Дальнейшие исследования можно проводить для других типов и форматов документов. Обобщение извлечения иерархической структуры для других типов документов (например, законы или научные статьи) является более трудоёмкой задачей.

## Список литературы

- [1] 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.
- [2] Wexler, Michael C. Structure extraction on electronic documents. — 2001. — 2. — US Patent 6,298,357.
- [3] Pembe, F Canan. A Tree Learning Approach to Web Document Sectional Hierarchy Extraction. / F Canan Pembe, Tunga Güngör // ICAART (1). — 2010. — Pp. 447–450.
- [4] 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.
- [5] Namboodiri, Anoop M. Document structure and layout analysis / Anoop M Namboodiri, Anil K Jain // Digital Document Processing. — Springer, 2007. — Pp. 29–48.
- [6] 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.
- [7] Извлечение логической структуры из сканированных документов / Анастасия Олеговна Богатенкова, Илья Сергеевич Козлов, Оксана Владимировна Беляева, Андрей Игоревич Перминов // Труды Института системного программирования РАН. — 2020. — Vol. 32, no. 4.
- [8] Goldfarb, Charles F. The SGML handbook / Charles F Goldfarb. — Oxford University Press, 1990.
- [9] Bray, Tim. Extensible markup language (XML) 1.0. — 2000.
- [10] Walsh, Norman. DocBook: the definitive guide / Norman Walsh, Leonard Muellner. — "O'Reilly Media, Inc. 1999.
- [11] Day, Don. Introduction to the Darwin Information Typing Architecture / Don Day, Michael Priestley, David Schell // IBM corporation. — 2005.

- [12] *Musciano, Chuck.* HTML & XHTML: The Definitive Guide: The Definitive Guide / Chuck Musciano, Bill Kennedy. — "O'Reilly Media, Inc. 2002.
- [13] *Belaid, Abdel.* XML data representation in document image analysis / Abdel Belaid, Yves Rangoni, Ingrid Falk // Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) / IEEE. — Vol. 1. — 2007. — Pp. 78–82.
- [14] Office Open XML File Formats — Fundamentals and Markup Language Reference. — 2016.
- [15] *Appelt, Wolfgang.* The formal specification of the ISO open document architecture (ODA) standard / Wolfgang Appelt, Nik Tetteh-Lartey // *The Computer Journal*. — 1993. — Vol. 36, no. 3. — Pp. 269–279.
- [16] Comparison of JSON and XML data interchange formats: a case study. / Nurzhan Nurseitov, Michael Paulson, Randall Reynolds, Clemente Izurieta // *Caine*. — 2009. — Vol. 9. — Pp. 157–162.
- [17] *Ben-Kiki, Oren.* Yaml ain't markup language (yaml<sup>TM</sup>) version 1.1 / Oren Ben-Kiki, Clark Evans, Brian Ingerson // *Working Draft 2008-05*. — 2009. — Vol. 11.
- [18] *Lamport, Leslie.* LATEX: a document preparation system: user's guide and reference manual / Leslie Lamport. — Addison-wesley, 1994.
- [19] PDF reference and Adobe extensions to the PDF specification.
- [20] *Rahman, Muhammad Mahbubur.* Unfolding the Structure of a Document using Deep Learning / Muhammad Mahbubur Rahman, Tim Finin // *arXiv preprint arXiv:1910.03678*. — 2019.
- [21] Scikit-learn Documentation. <https://scikit-learn.org>.
- [22] XGBoost Documentation. [https://xgboost.readthedocs.io/en/latest/python/python\\_api.html#module-xgboost.sklearn](https://xgboost.readthedocs.io/en/latest/python/python_api.html#module-xgboost.sklearn).
- [23] Python-docx Documentation. <https://python-docx.readthedocs.io/en/latest/>.
- [24] Docx2python Documentation. <https://pypi.org/project/docx2python/>.

- [25] Pydocx Documentation. <https://github.com/CenterForOpenScience/pydocx>.
- [26] Официальный сайт единой информационной системы в сфере закупок в информационно-телекоммуникационной сети Интернет. <https://zakupki.gov.ru>.
- [27] ImageClassifier. <https://github.com/dronperminov/ImageClassifier>.
- [28] Treelib documentation. <https://treelib.readthedocs.io/en/latest/>.
- [29] Github repository with code. <https://github.com/NastyBoget/DOCXParser>.

## Приложение 1

Название	Основная цель	Основан на	Физическая структура	Логическая структура	Программная поддержка	Сложность	Читаемость
SGML	Для совместного использования машинно-читаемых документов в больших проектах	GML	Не предназначен для хранения физической структуры	Создан для этого физической структуры	Есть формат (но устарел)	Спецификация одна, но сложна в освоении	Читаемо, но всегда удобно
DAFS	Для результатов распознавания изображений документов	SGML	Дерево физической структуры	Дерево логической структуры	Сейчас используется, скорее всего нет	Много сложных понятий	Вряд ли легко воспринимать, но примеров представления не найдено
XML	Для создания и обработки документов	SGML	Можно использовать	Можно использовать	Есть, общирная	Средняя сложность	Достаточно удобно читать

DocBook	Для технической документации	SGML / XML	Не предоставляет информацию о визуальном представлении документа	Представляет возможность описания логической структуры документа	Есть	Стандарт большой сложный в освоении	Читаемо, есть стредством для визуализации
DITA	Для эффективного повторного использования контента	XML	Как выглядит документ контролирующий выходные форматы	Для языческой структуры используется семантическое тегирование	Есть	цифрические понятия, которые нужно знать для работы	Содержимое разбито на части (может быть неудобно), визуализация в выходных форматах
HTML, XHTML, HTML5	Язык структурирования и представления содержимого всемирной паутины	SGML / XML	Есть обширные возможности для описания того, как выглядит документ	Есть возможность описания для описания того, как выглядит документ	Относительно несложные и известные форматы	Читаемо и визуализируется в браузерах	

ALTO + METS	Для хранения результатов OCR	XML	Описывается в ALTO	Описывается в METS	Есть	Сложно	Есть инструменты визуализации
TEI	Для представления семантической и логической структуры документов	XML	Существуют несколько тегов для описания визуальных особенностей текста	Создан для отображения логической структуры визуальных особенностей текста	Обширный список различных инструментов для работы с форматом	Содержит обширный набор тегов инструментов для работы с форматом	Есть инструменты для визуализации, формат читаем
ODA	Для описания структурированных данных объектного типа	—	Есть	Есть	Из-за сложности стандарта существует мало инструментов	Огромный и сложный стандарт	Сложно читать, инструментов визуализации мало

Office Open XML	Для хранения электронных документов пакетов офисных приложений	zip-архив, содер-жаций текст в виде XML, графику и другие данные	Обширные возмож-ности для описания физической структуры	Практически нет	Есть приложения и библиотеки для работы, однако полностью обработать документ сложно	Очень большая документация	Читаемо в ях, в xml структура сложная
JSON	Для сериализации сложных структур, в веб-приложениях - для обмена данными	Расширен из JavaScript	Существует возможность описания	Есть	Обширная программная поддержка	Простой в освоении формат	Не удобен для чтения
YAML	Язык разметки, удобный для представле-ния структур данных в различных ЯП	Расширен из XML, Perl и JSON	Существует возможность описания	Есть	Библиотеки для большого числа наиболее популярных ЯП	Относительно несложный	Читаемый

TeX	Система компьютерной вёрстки для документов для компьютерной типографии	—	Есть	Есть	Существует множество расширений программы вёрстки TeX	Относительно несложен в освоении, фокусировка на содержимое документа	Хорошо читается, можно преобразовать в другие форматы для визуализации
PDF	Для представления полиграфической продукции в электронном виде	Расширен	Есть	Нет	Достаточно бедные инструменты для работы	Большая спецификация	Хорошо читается практически на любой машине

## Приложение 2

### 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 15, 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. 15: Пример задания

**Титульный лист (титульник)** – есть не во всех ТЗ, но если есть, то это первая страница документа (примеры на рисунке 16). Содержит название, информацию о заказчике/исполнителе и так далее. Все параграфы титульника надо отметить как **title**.

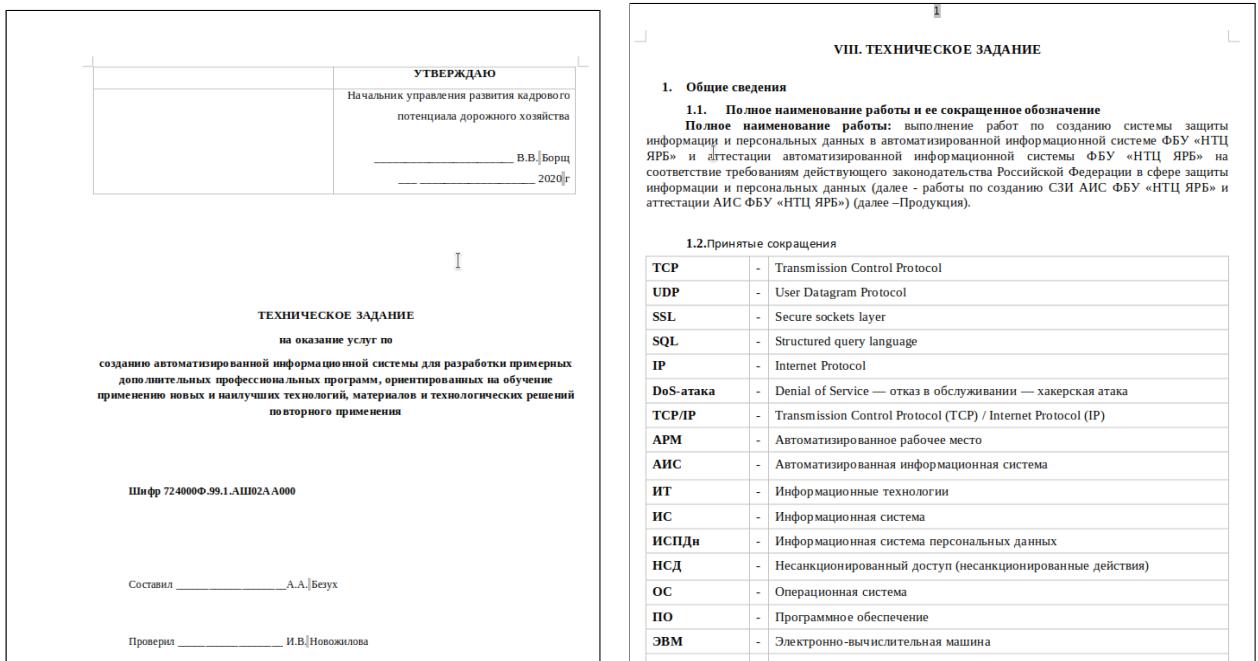


Рис. 16: Пример ТЗ с титульным листом (слева) и без титульного листа (справа)

Если в ТЗ нет титульного листа, но есть «шапка», то её тоже надо отметить как **title** (на рисунке 17 **title** — это все параграфы до «1. Описание продукции»).

**Оглавление** – есть не в каждом ТЗ, но если есть, то похоже на то, что изображено на рисунке 18. Каждый параграф оглавления надо выделить в класс **toc** (включая слово “Содержание”).

**Часть документа (part)** – часть документа может выделяться как текстом, так и оформлением. Выделение текстом означает, что параграф содержит слово раздел/подраздел или что-то подобное. Например, «Раздел 2 Основные положения». Часть может выделяться оформлением: отступом, жирностью, размером текста. При этом должен присутствовать номер. Текст **части (part)** существенно отличается от основного текста, часть выделена (см. рисунок 19).

**Элемент списка (item)** – начинается с числа или нескольких чисел, разделённых точкой, или специального символа (тире или чёрный кружок). На рисунке 20 элемента-

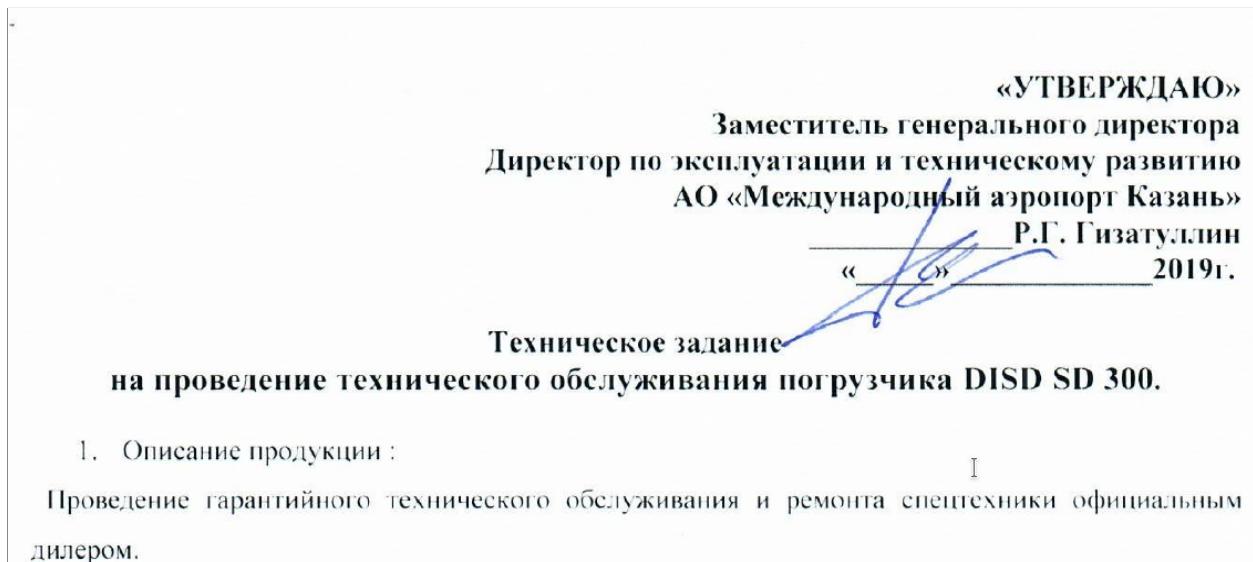


Рис. 17: Пример ТЗ с «шапкой»

ми списка являются параграфы, выделенные красной рамкой (остальные невыделенные строки — это raw\_text).

**Просто текст** – это обычный текст, то есть всё остальное. На рисунке 21 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. 18: Пример оглавления

- оказание услуги по сопровождению системы защиты персональных данных.

**1.1. Требование по соответствию оказываемых услуг действующему законодательству Российской Федерации**

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.

**9. Общие требования к лицензионному системному и прикладному программному обеспечению, средствам защиты информации, их качеству, техническим характеристикам и безопасности:**

9.1. Требования к средству защиты информации от несанкционированного Secret Net 7 (сетевой режим работ).

Должно осуществлять:

- защиту серверов и рабочих станций от НСД;

Рис. 19: Примеры части

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.

При макетировании и тестировании системы защиты информации информационной системы в том числе осуществляются:

- проверка работоспособности и совместимости выбранных средств защиты информации с информационными технологиями и техническими средствами;

**9.2. Требования к программно-аппаратному комплексу "Соболь". Версия 3.0. или эквивалент.**

Должен осуществлять:

Рис. 20: 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. 21: Пример обычного текста

# Приложение 3

## 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:

## VIII. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

### 1. Общие сведения

#### 1.1. Полное наименование работы и ее сокращенное обозначение

**Полное наименование работы:** выполнение работ по созданию системы защиты информации и персональных данных в автоматизированной информационной системе ФБУ «НТЦ ЯРБ» и аттестации автоматизированной информационной системы ФБУ «НТЦ ЯРБ» на соответствие требованиям действующего законодательства Российской Федерации в сфере защиты информации и персональных данных (далее - работы по созданию СЗИ АИС ФБУ «НТЦ ЯРБ» и аттестации АИС ФБУ «НТЦ ЯРБ») (далее –Продукция).

Рис. 22: Пример задания

Необходимо сравнить два параграфа, обведённые в красную и синюю рамки (рис. 22).

### 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.

- Федеральный закон от 27 июля 2006 г. № 149-ФЗ «Об информации, информационных технологиях и о защите информации»;
- Федеральный закон от 28 декабря 2010 г. № 390-ФЗ «О безопасности»;
- Федеральный закон от 27 декабря 2002 г. № 184-ФЗ «О техническом регулировании»;
- Федеральный закон от 27 июля 2006 г. № 152-ФЗ «О персональных данных»;
- Федеральный закон от 04 мая 2011 г. № 99-ФЗ «О лицензировании отдельных видов деятельности»;

Рис. 23: Пример равных параграфов: элементы списка с одинаковым оформлением

В примере на рисунке 23 параграфы – это элементы списка с одинаковым оформлением.

<b>1.5 Срок выполнения работ</b>
<b>Срок начала выполнения работ:</b> с даты заключения Договора.
<b>Срок окончания выполнения работ:</b> 120 календарных дней с даты заключения Договора.
<b>1.1. Требование по соответствию оказываемых услуг действующему законодательству Российской Федерации</b>
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.
<b>1.2. Услуги по формированию требований к защите информации, содержащейся в информационной системе</b>
Формирование требований к защите информации, содержащейся в информационной системе включает:
- классификацию информационной системы по требованиям защиты информации;

Рис. 24: Примеры равных параграфов: текстовые строки с одинаковым оформлением

В примерах на рисунке 24 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 визуально и логически включает в себя второй параграф.
2. Первый параграф визуально «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.

документов.

Разработка системы защиты информации информационной системы осуществляется в соответствии с техническим заданием на создание системы защиты информации информационной системы и включает:

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

Рис. 25: Пример, в котором первый параграф больше второго (структурная вложенность элемента списка в параграф)

В примере на рис. 25 элемент списка (второй параграф, обведённый в синюю рамку) структурно вложен в параграф, обведённый в красную рамку.

## 2. Технические, функциональные и эксплуатационные требования к СИСТЕМЕ

### 2.1. Общие требования

2.1.1. СИСТЕМА должна соответствовать требованиям к безопасности, качеству, техническим характеристикам, функциональным характеристикам (потребительским свойствам) товара, установленные заказчиком и предусмотренные техническими оставляемого товара, выполняемой работы, оказываемой услуги потребностям заказчика, в том числе:

2.1.1.1. ГОСТ 34.601-90 «АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ. СТАДИИ СОЗДАНИЯ»;

2.1.1.2. ГОСТ 34.003-90 «АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ. ТЕРМИНЫ И ОПРЕДЕЛЕНИЯ»;

Рис. 26: Пример, в котором первый параграф больше второго (структурная вложенность элементов списка)

В примере на рис. 26 элемент списка в синей рамке структурно вложен в другой элемент списка в красной рамке (по нумерации).

В примерах на рисунке 27 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 28 структурный элемент второго параграфа в синей рамке

## ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на оказание услуг по разработке и внедрению автоматизированной системы  
«Адресная социальная помощь»

ОГБУ «Многофункциональный центр предоставления государственных и  
муниципальных услуг Еврейской автономной области»

### **Требования к объёму и характеристикам оказываемых услуг.**

#### **1. Объем оказываемых услуг:**

Состав и содержание услуг по созданию Системы защиты клиентского сегмента АИС МФЦ НО МФЦ на 18 окон (клиентских мест):

Рис. 27: Примеры, в которых первый параграф больше второго (оформление)

#### **1.3. Плановые сроки начала и окончания работ**

Начало работ – дата подписания Контракта.

Окончание работ –31.07.2020 г.

#### **1.4. Перечень НПА, на основании которых выполняются работы**

- Жилищный кодекс Российской Федерации от 29.12.2004 г. № 188-ФЗ;

Рис. 28: Пример, в котором первый параграф меньше второго (структурная вложенность)

важнее первого в красной рамке.

В примере на рисунке 29 после вложенного списка в красной рамке начинается текст второго параграфа в синей рамке.

В примере на рисунке 30 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 31.

**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. 29: 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. 30: 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. 31: 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.