Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ __Радиотехнический_____

КАФЕДРА ____Системы обработки информации и управления_____

РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ

НА ТЕМУ:

«Модуль формирования концептуальных карт на основе текста для русского языка»

Студент ___РТ5-81___ _____ __**Г.А. Евсеев**_____
(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР _____ __**А.А. Максаков**___
(Подпись, дата) (И.О.Фамилия)

Консультант _____ __**Ю.Е. Гапанюк** ___
(Подпись, дата) (И.О.Фамилия)

Нормоконтролер _____ __**Ю.Н. Кротов**___
(Подпись, дата) (И.О.Фамилия)

2023 г.

АННОТАЦИЯ

Расчётно-пояснительная записка квалификационной работы бакалавра содержит 53 страницы. С приложениями объем составляет 73 страницы. Работа включает в себя 12 таблиц и 13 иллюстраций. В процессе выполнения было использовано 24 источника.

Цель работы заключается в создании модуля формирования концептуальных карт на основе текста для русского языка. Разрабатываемый модуль должен включать в себя алгоритмы синтаксической и морфологической разметки, лемматизации русскоязычного текста и визуализации полученной концептуальной карты.

Разработка модуля производилась на основании документа «Техническое задание», утвержденного руководителем выпускной работы.

В процессе выполнения квалификационной работы бакалавра были реализованы все требования и модуль соответствует всем сформулированным требованиям. Была составлена расчетно-пояснительная записка.

Пояснительная записка содержит 3 приложения.

СОДЕРЖАНИЕ

СПИСОК ОБОЗНАЧЕНИЙ И СОКРАЩЕНИЙ

Словоформа – Слово в узком смысле, то есть обладающая признаками слова цепочка фонем.

Граммема – Грамматическое значение, понимаемое как один из элементов грамматической категории; различные граммемы одной категории исключают друг друга и не могут быть выражены вместе.

Корпус – подобранная и обработанная по определенным правилам совокупность текстов

Токенизация – по-другому сегментация – это процесс разделения письменного языка на предложения-компоненты.

Токен – объект, создающийся из лексемы в процессе лексического анализа (токенизации).

NLP, Natural Language Processing – обработка естественного языка – подраздел информатики и AI, посвященный тому, как компьютеры анализируют естественные языки.

CMM, Concept Map Mining – Подход к автоматической генерации концептуальных карт из текста.

Именная группа – лингвистический термин, определяющий группу имен в форме словосочетаний, составляющий компонент иерархической структуры предложения, которые обладают синтаксическими свойствами существительного.

Глагольная группа – синтаксическая единица, состоящая из по меньшей мере одного глагола и его зависимых, таких как дополнения, комплементы и другие модификаторы.

Клауза – составляющая вершиной которой является глагол либо связка или элемент, которая играет роль глагола в предложении.

POS-tagging, частеречная разметка – этап автоматической обработки текста, задачей которого является определение части речи и грамматических характеристик слов в тексте с приписыванием им соответствующих тегов.

NER, Named Entity Recognition – подзадача извлечения информации, которая ищет и выделяет именованные сущности, упомянутые в неструктурированном тексте, и выделяет их в заранее выделенные категории.

IR, Information Retrieval – процесс поиска неструктурированной документальной информации, удовлетворяющей информационные потребности.

IE, Information Extraction – задача автоматического извлечения структурированных данных из неструктурированных или слабоструктурированных документов.

CE, Concept Extraction - задача автоматического извлечения концептов из текста.

RE, Relationship Extraction – задача автоматического извлечения связей между двумя концептами из текста, а так же определения типа связи.

TF-IDF – статистическая мера, используемая для оценки важности слова в контексте документа, являющегося частью коллекции документов или корпуса.

Эмбединг – представление слова в виде многомерного вектора. Кодирует слова таким образом, что слова похожие по значению расположены ближе друг другу в векторном пространстве.

ВВЕДЕНИЕ

В настоящее время наблюдается рост использования концептуальных карт. Самое частое применение концептуальных карт – сбор и представление уже полученных знаний в форме, удобной для восприятия человеком. Более того, концептуальные карты известны как эффективный инструмент для организации и навигации по большим объемам информации.

Так же концептуальные карты могут быть использованы как средство для представления плана обучения, где концепты будут представлять набор целей, а связи между ними – средствами для достижения целей. Зачастую сложно с нуля составить карту по выбранной теме. Каркас схемы, предоставленный экспертом в интересующей человека области, может сильно облегчить задачу, но при индивидуальном изучении зачастую тяжело найти такого человека. Поэтому информационная система, которая заменит эксперта и представит каркас концептуальной карты может быть очень полезна в такой ситуации.

Создание модуля автоматической генерации концептуальной карты на естественном языке позволяет решить сразу несколько проблем. Она позволит заменить эксперта и получать концептуальные карты без помощи человека, которые потом возможно будет использовать, для последующей автоматической генерации текстов. Так же такая концептуальная карта помогает систематизировать знания, облегчает восприятие текста читателем, упрощает синтаксическую и морфологическую проверку текста.

Квалификационная работа на тему «Модуль формирования концептуальных карт на основе текста для русского языка» посвящена разработке алгоритма получения концептуальных карт из неструктурированного текста на русском языке.

Разработка модуля проводилась в рамках кафедральных исследований по разбору текста, формирования концептуальной карты русского языка и последующей генерации текстов на основе полученной концептуальной карты.

1 ПОСТАНОВКА ЗАДАЧ РАЗРАБОТКИ

1.1 Общетехническое обоснование разработки

1.1.1 Постановка задачи проектирования

Разработке подлежит модуль формирования концептуальных карт из текста для русского языка. Его цель – автоматическая генерация концептуальных карт. В задачу формирования концептуальных карт входит: очистка текста от ненужных конструкций и последующее разбиение текста на предложения, морфологический и синтаксический анализ предложений, выделение именных конструкций из предложений, выделение глагольных конструкций из предложений, определение триплетов вида «концепт-связь-концепт», лемматизация полученных триплетов, сборка и визуализация концептуальной карты, на основе полученных триплетов.

Концептуальная карта – это графический инструмент, использующийся для структурирования и представления знаний. Она включает в себя концепты, которые чаще всего представлены существительными и именными группами и связями между ними, указываемые линией, соединяющей два концепта. Обозначение линии глаголом или глагольной группой создает цепочку концепт-связь-концепт, которую можно прочитать как предложение. Эта цепочка называется утверждением [1]. Пример концептуальной карты представлен на рисунке 1.

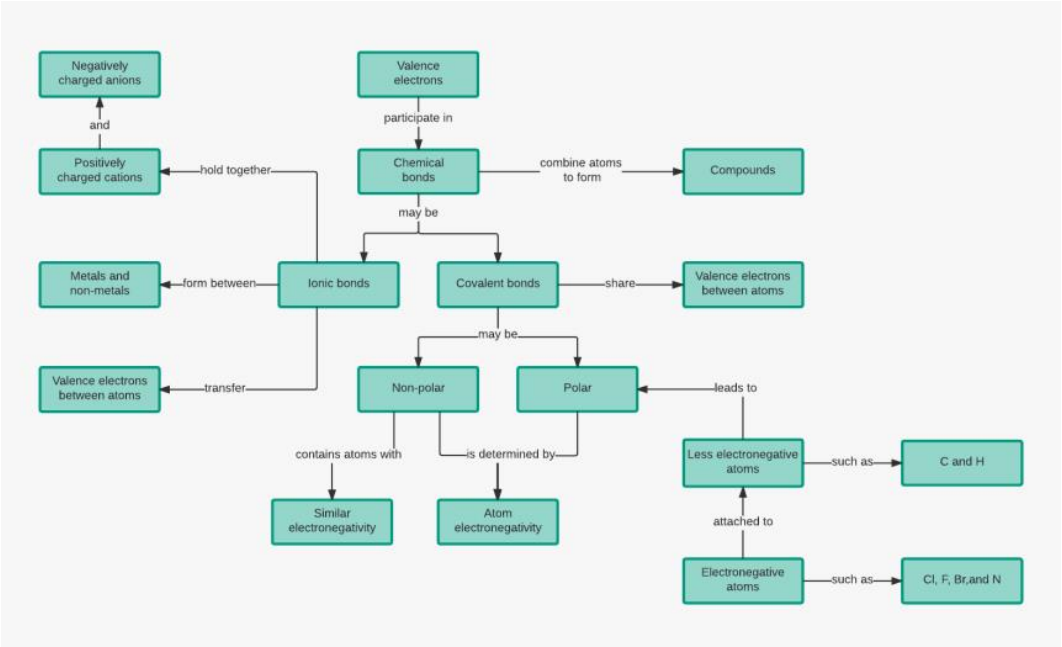


Рисунок 1 – Пример концептуальной карты

Морфологический анализ в лингвистике представляет собой определение морфологических характеристик слова, где каждая характеристика – это граммема слова. Различные граммемы одной категории исключают друг друга и не могут быть выражены вместе. Так, например, есть граммемы множественного и единственного числа, 1-го, 2-го и 3-го лица, совершенного и несовершенного вида [2]. Так же характеристикой слова является его лемма – словарная форма слова. Пример такой морфологической разметки представлен на рисунке 2.

МАМА	ПОШЛА	КУПИТЬ	ПРОДУКТЫ
Animacy=Anim Case=Nom Gender=Fem Number=Sing мама NOUN	Aspect=Perf Gender=Fem Mood=Ind Number=Sing Tense=Past VerbForm=Fin Voice=Act пойти VERB	Aspect=Perf VerbForm=Inf Voice=Act купить VERB	Animacy=Inan Case=Acc Gender=Masc Number=Plur продукт NOUN

Рисунок 2 – Пример морфологический разметки простого предложения

Синтаксический анализ – процесс составления линейной последовательности лексем (токенов) естественного языка с его грамматикой.

Результатом разбора обычно бывает дерево разбора или по-другому – синтаксическое дерево. Для синтаксического анализа используют синтаксические парсеры, которые находят в тексте и выделяют синтаксические конструкции, проверяют правильной каждой полученной конструкции и представляют всю информацию в виде синтаксического дерева. На основе такого разбора можно выполнять задачи перевода текста, извлечения текста, поиска информации в тексте.

В ходе выполнения работы были определены следующие цели:

- Исследование предметной области формирования концептуальных карт.
- Исследование предметной области обработки естественного языка.
- Определение функциональных требований.
- Изучить и провести сравнительные анализ имеющихся на данный момент решений.
- Выбрать модель машинного обучения, улучшающую предсказания.
- Программная реализация модуля.
- Провести тестирование модуля.

Успешно решив все подзадачи, мы получим эффективный модуль генерации концептуальных карт для текстов на русском языке.

1.1.2 Описание предметной области

Предметной областью является компьютерная лингвистика, а также теория автоматизированной генерации концептуальных карт – Concept map mining.

Компьютерная лингвистика, а конкретнее обработка естественных языков (Natural language processing) ставит перед собой задачи изучения анализа, обработки, и последующей генерации текстов на естественных языках компьютером. Понимание естественного языка считается полной AI-задачей,

потому как распознавание языка требует огромных знаний системы об окружающем мире.

Для решения задач обработки естественного языка, таких как распознавание речи, анализ текста, генерация текста, синтез речи и других, сначала нужно провести пред лингвистическую обработку текста, а именно: очистить текст от ненужных конструкций (ссылки, хештеги, ссылки на литературу и другие), разбить текст на абзацы, после разбить абзацы на предложения. Затем уже на предложениях проводится морфологический анализ, синтаксический анализ, и решение последующих задач.

Concept map mining представляет собой процесс извлечения информации из одного или более документов для автоматического создания концептуальной карты. Созданная карта является обобщенной сводной исходного текста.

С точки зрения CMM, документ может быть представлен как набор концептов и связей.

CMM может быть как полуавтоматическим, так и полностью автоматизированным. В полуавтоматическом режиме система находит и предлагает некоторые элементы карты, и человек должен закончить карту вручную, используя предоставленную информацию. В автоматической реализации, на выходе пользователь получает конечную концептуальную карту исходя из входного текста.

1.1.3 Функциональные задачи системы

Функциональные задачи должны быть сформированы исходя из целей, которые были поставлены перед разрабатываемым модулем.

Функциональная задача – это реализация функций программного модуля.

Доступ к модулю должен быть возможен как через веб-интерфейс, так и с помощью командной строки.

Разрабатываемый модуль должен выполнять следующие функции:

- Обработка входного текста;

- Разбиение текста на предложения;
- Выделения глагольных конструкций из предложений;
- Выделение именных конструкций из предложений;
- Составление триплетов вида “концепт-отношение-концепт” для каждого предложения, на основе полученных именных и глагольных форм;
- Лемматизация триплетов;
- Визуализация предложений на основе меток зависимости;
- Составление и визуализация концептуальной карты, на основе суммаризации полученных триплетов.

2 КОНСТРУКТОРСКО-ТЕХНОЛОГИЧЕСКАЯ ЧАСТЬ

2.1 Выбор программных средств

При выборе операционной системы рассматривалось два варианта: Windows OS и Linux. Windows – самая популярная операционная система на рынке в данный момент, обладающая удобный пользовательским интерфейсом, из-за чего является более простой в использовании, нежели Linux подобные системы. Так же Windows имеет более широкий выбор средств для разработки и имеет поддержку большинства программных продуктов, а для установки некоторых решений на Linux, придется прибегнуть к установке дополнительных компонентов. Поэтому выбор был сделан в сторону Windows.

Следующим шагом будет выбор языка программирования, который будет наиболее удобен для разработки модуля формирования концептуальных карт на основе текста для русского языка.

Наиболее удобным вариантом будет язык программирования Python3, так как он обладает множеством решений, которые можно будет использовать при разработке модуля, что упростит задачу. Большинство решений задач обработки естественного языка написаны именно на языке программирования Python3.

Python – один из самых популярных языков программирования, который активно используется при решении задач обработки естественного языка. Код, написанный на этом языке, легко читаем, некоторые функции занимают меньше строк кода, по сравнению с аналогами, из-за простоты синтаксиса, что увеличивает скорость разработки программных продуктов на этом языке программирования.

Все исследования и сравнения, а также сам модуль были написаны при помощи Jupyter Notebook.

Список программных библиотек и их версий, которые были использованы в работе, представлены в таблице 1.

Библиотека Networkx – является библиотекой для визуализации графов, очень проста в освоении, но и имеет достаточно ограниченный функционал. Для реализации, поставленной задачи данного функционала, будет достаточно, так что выбор был сделан в сторону простоты использования.

Библиотека re нужна для того, чтобы с помощью регулярных выражений проводить очистку текста, а также разбиение текста на предложения.

ru-core-news-sm – модель пред обученная для задач NLP на русском языке.

Библиотека textacy – удобная обертка для библиотеки SpaCy для работы с текстом, имеет возможность с помощью шаблонов находить разные конструкции в тексте, проводить сравнение словоформ по их токенами, и множество других разнообразных задач по работе с текстом.

Таблица 1. Список библиотек и версий

Название библиотеки	**Версия**
import_ipynb	0.1.4
Networkx	3.1
matplotlib	3.7.1
numpy	1.24.3
ru-core-news-sm	3.3.0
spacy	3.5.0
textacy	0.13.0
Pymorphy2	0.9.2
re	3.11.4

Так как нету графического интерфейса это упрощает выполнение работы и освобождает от поиска фреймворка для разработки веб-приложений, таких как Streamlit.

2.2 Выбор аппаратных средств

Аппаратное обеспечение – электронные и механические части вычислительного устройства, входящие в состав системы или сети, исключая программное обеспечение и данные. То, как быстро будет обрабатываться запросы, и как быстро будет выполнятся программа напрямую зависит от аппаратных характеристик вычислительного устройства. Для обучения модели, и для тестирования потребуется большой объем оперативной памяти. Частота процесса и кол-во ядер напрямую повлияет на быстродействие программы.

Минимальные системные требования для полноценной работы клиентской части:

- Процессор с тактовой частотой 1200ГГц;
- Оперативная память не менее 16 ГБ;
- Видеокарта – GeForce GTX 1070 или выше;
- Видеоадаптеры и мониторы способные обеспечить графический режим 1024*768 точек;
- Жёсткий диск объемом 15 ГБ;
- Манипулятор «мышь» или другое указывающее устройство;
- Клавиатура.

2.3 Разработка программного изделия

На основе функциональных требований был разработан модуль формирования концептуальных карт из текста для русского языка. Он принимает на вход необработанный неструктурированный текст, и на его основе генерирует концептуальную карту данного текста.

Для реализации основной части была использована библиотека SpaCy, а также вспомогательная библиотека Textacy, которая является оберткой библиотеки SpaCy для работы с текстом.

Ниже будет описан полный набор алгоритмов написанные в реализованном модуле. Графическое описание алгоритма представлено в приложении А.1

2.3.1 Очистка текста

На вход в программу подается неструктурированный необработанный текст, который может содержать лишние конструкции, такие как ссылки на сторонние сайты, ссылки в тексте на использованную литературу, хештеги, упоминания. Такие конструкции являются избыточными и на этапе лингвистической пред обработки от них нужно избавиться.

Ниже представлена функция очистки текста. Результаты выполнения функции представлены на рисунке 3.

```
class text_treatment:
def __init__(self, text):
    self.text = text
    self.clean_sent = []
def clean(self):
    #removing paragraph numbers
    self.text = re.sub('[0-9]+.\t','',str(self.text))
    #removing new line characters
    self.text = re.sub('\n ','',str(self.text))
    self.text = re.sub('\n',' ',str(self.text))
    #removing apostrophes
    self.text = re.sub("'s",'',str(self.text))
    #removing hyphens
    self.text = re.sub("-",' ',str(self.text))
    self.text = re.sub(" – ",' ',str(self.text))
    #removing quotation marks
    self.text = re.sub('\"','',str(self.text))
```

```
        #removing any reference to outside text
        self.text = re.sub("[\(\[].*?[\)\]]", "",
str(self.text))
        #removing urls
        self.text = re.sub(r'http\S+', "",
str(self.text))
        #removing hashtags
        self.text = re.sub('#',"", str(self.text))
```

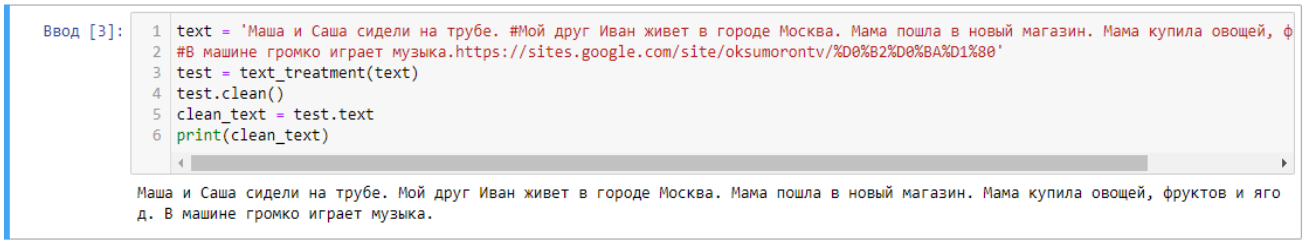


Рисунок 3 – Результат выполнения функции очистки текста

2.3.2 Разбиение текста на предложения

После очистки текста, для последующей токенизации, а также проведения морфологического и синтаксического анализа, его требуется разбить на предложения. После каждое предложение токенизируется – выделяются базовые элементы языка. Элементами чаще всего являются слова или фразы разделенные не буквенно-цифровыми токенами, такие как пробел или пунктуационный знак. Все слова так же преобразуются в нижний регистр. Есть исключения, такие как аббревиатуры, знаки переноса, цифры и некоторые другие сущности. Они тоже будут переведены в нижний регистр для удобства обработки. Для токенизации был использован конвейер, взятый из библиотеки SpaCy, обученный на модели ru_core_news_sm. Для обучения данной модели был использован корпус новостных статей Lenta.ru, в котором собраны новостные статьи с 1999 по 2019 год [3]. Данный корпус находится в открытом доступе. На этом же этапе конвейер произведет морфологический анализ и синтаксический анализ предложений. Морфологический анализ производится

при помощи анализатора PyMorphy2. PyMorphy2 – морфологический парсер, который содержит открытый исходный код, и предоставляет все функции полного морфологического анализа и синтеза словоформ [4].

Парсер основывается на словарной морфологии и использует данные проекта OpenCorpora [5]. Данный словарь содержит в себе более 391 тыс. лемм, является открытым продуктом и постоянно пополняется множеством пользователей.

Во время морфологического синтеза, с помощью исходной словоформы и ее тегам осуществляется поиск леммы слова, а затем перебираются все возможные комбинации вида <окончание, теги> в найденной лексеме, до тех пор, пока не найдется такая пара, которая соответствует заданным морфологическим тегам. После этого у нормальной формы удаляется окончание, а найденное окончание приписывается к псевдооснове.

Для анализа ненайденных в словаре слов в PyMorphy2 используется ряд методов, которые применяются друг за другом в определенной последовательности. Первый метод – из слова удаляется префикс, из полного набора известных префиксов и ищется полученный остаток слова в словаре и если остаток находится в словаре, то удаленный префикс записывается к результату разбора. Если метод не дал результата и остаток слова не был найден в словаре, то тоже самое проводится для префикса слова длиной от 1 до 5, в независимости известен такой префикс или он является неизвестным. Если этот метод тоже не сработал, то словоформа разбирается по окончанию. Для этого используется набор всех известных окончаний, встречающихся в данном словаре. По мере выполнения метода из набора удаляются неподходящие окончания и разборы. Такой анализ по окончанию имеет схожий алгоритм работы с алгоритмом, который использован в процессоре AOT.

Ниже представлена функция, выполняющая разбиение текста на предложения, а также токенизацию предложений. Результат выполнения функции представлен на рисунке 4.

```
def sentences(self):
    # split sentences and questions
        self.text = re.split('(?<!\..)[.?!]\s+',
self.text)
        for sent in self.text:
            self.clean_sent.append(sent)
        for sent in self.clean_sent:
            sent = nlp(sent)
```

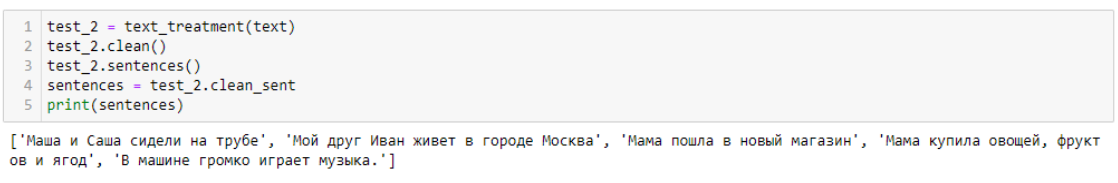


Рисунок 4 – Результат выполнения функции разбиения текста на предложения

2.3.3 Выделение глагольных групп из предложений

Затем после этапа токенизации, мы можем выделить глагольные группы из предложений, при помощи их dependency меток. Dependency метка слова показывает связь между двумя фразами. Если слово является дополнением смысла другого слова, то оно будет соединено меткой зависимости. Для выделения полной глагольной группы требуется собрать набор шаблонов, по которым будет осуществляться поиск и выделение полных глагольных групп из предложения. Глагольная группа всегда состоит из корня предложения (root), а также набора модификаторов, которые могут дополнять смысл: adp – adposition, adv – adverb, advmod – adverbial modifier, adcl – adverbial clause modifier и других. Полный набор унифицированных меток представлен в таблице 2.

Таблица 2. Список меток зависимости и их расшифровка

Метка	**Расшифровка метки**
nsubj	Nominal subject
csubj	Clausal subject
nsubj:pass	Passive clause subject
obj	Objective
iobj	Indirect objects
ccomp	Clausal dependency of verbs
xcomp	Open clausal dependency
obl	Prepositional phrase
vocative	Participant addressed in dialogue
exlp	Expletive or pleonastic nominal
acl:relcl	Relational clause
nmod	Nominal modifier
appos	Appositive
compound	Flat compound relation
amod	Adjectival modifier
det	Determiner
aux	Auxiliary
cop	Copula marker
case	Preposition and postposition
cc	Coordinate conjunction
mark	Subordinate conjunction
punct	Punctuation mark

Продолжение таблицы 2

Метка	Расшифровка метки
root	Root of the sentence
orphan	Orphan relation
list	Chains of items
fixed	Token in the multiword expression
flat	Flat relation
goeswith	Element written separately in not well edited texts
dep	Unspecified dependency

Все Dependency метки унифицированы и собраны в проекте Universal Dependencies. В данном проекте собрано более 100 корпусов для более 70 языков, включая русский, которые были приведены к одному формату. Именно благодаря данному проекту появилась возможность лингвистической согласованности аннотаций и стали расширятся языковые модели [6].

Алгоритм работает в 2 этапа, сначала находится корень предложения, который будет являться каркасом глагольной группы, он имеет метку зависимости root. Затем при помощи собранных шаблонов собираются все возможные варианты глагольных групп, которые могут получится из полученного каркаса с использованием всем модификаторов в окружности от корня. Из всех возможных вариантов выбирается наиболее полный, и именно он будет являться нашей глагольной группой, а также связью между двумя концептами в предложении. Такой поиск осуществляется для каждого предложения и на выходе получается полный набор достаточных глагольных групп. Графическая реализацию алгоритма изображена в приложении А.3.

Ниже представлены функции, выполняющие поиск глагольных групп. Результаты выполнения функций представлены на рисунке 5.

```
def contains_root(self, verb_phrase, root):
        vp_start = verb_phrase.start
```

```
            vp_end = verb_phrase.end
            if (root.i >= vp_start and root.i <= vp_end):
                return True
            else:
                return False

    def find_root_of_sentence(self, doc):
            root_token = None
            for token in doc:
                if (token.dep_ == "ROOT"):
                    root_token = token
            return root_token

    def get_verb_phrases(self, doc):
            root = self.find_root_of_sentence(doc)
            verb_phrases =
textacy.extract.matches.token_matches(doc,
self.verb_patterns)
            new_vps = []
            for verb_phrase in verb_phrases:
                if (self.contains_root(verb_phrase, root)):
                    new_vps.append(verb_phrase)
            return new_vps

    def longer_verb_phrase(self, verb_phrases):
            longest_length = 0
            longest_verb_phrase = None
            for verb_phrase in verb_phrases:
                if len(verb_phrase) > longest_length:
                    longest_length = len(verb_phrase)
```

```
            longest_verb_phrase = verb_phrase
    return longest_verb_phrase
```

```
test_3 = text_triplets(sentences)
for sentence in sentences:
    sentence = nlp(sentence)
    vbs = test_3.get_verb_phrases(sentence)
    new_vbs = test_3.longer_verb_phrase(vbs)
    print(new_vbs)
```

```
сидели на
живет в
пошла в
купила
громко играет
```

Рисунок 5 – Результат выполнения функций выделения глагольных форм

2.3.4 Выделение именных групп из предложений

Затем, после выделения глагольных групп, мы выделяем именные группы, по аналогии с алгоритмом, по которому выделялись глагольные группы. Именная группа – лингвистический термин, определяющий группу имен в форме словосочетаний, составляющий компонент иерархической структуры предложения, которые обладают синтаксическими свойствами существительного и в которых имя существительное является вершиной группы, то есть главным словом. Например: шар, красный шар, большой красный шар.

Иногда к именным группам также группы с вершиной в виде местоимения, но чаще всего они будут обозначены как PRNP (pronoun phrase).

Обычно именные группы функционируют как объекты и субъекты глаголов и глагольных групп, и являются концептами в будущей концептуальной карте.

Так же как с выделением глагольных групп, выделение именных групп происходит в 2 этапа: на первом этапе мы выделяем каркас именной группы, который будет являться либо существительным, с унифицированной меткой

noun, либо имя собственное, отражающее имя, название страны, места, достопримечательности и т.д., с унифицированной меткой propn. На втором этапе мы собираем из каркаса все варианты именных групп, которые есть в предложении, также при помощи шаблонов. Затем выбирается наиболее полная именная группа, которая будет включать в себя все остальные группы, она и будет являться нашим концептом.

Зачастую, если глагольная группа в предложении одна, и она будет являться нашей связью, то именных групп, как минимум две – объект и субъект в предложении, но также бывают случаи перечисления объектов, или может быть несколько субъектов, которые выполняют действия над одним или более объектов в предложении. В таком случае будут собраны все именные группы, которые присутствуют в предложении. Это нужно для того, чтобы на конечном этапе – формировании и визуализации концептуальной карты, она максимально точно отражала суть исходного текста.

Ниже представлены функции, выполняющие поиск именных групп. Результат выполнения функций представлены на рисунке 6.

```
def find_noun_phrases(self, doc):
        noun_phrases =
textacy.extract.matches.token_matches(doc,self.noun_patte
rns)
        new_nph = []
        for noun_phrase in noun_phrases:
            new_nph.append(noun_phrase)
        return new_nph

    def longer_noun_phrase(self, noun_phrases):
        longest_length = 0
        noun_phrase_temp = None
        longest_noun_phrases = []
        for noun_phrase in noun_phrases:
```

```
            if noun_phrase_temp == None:
                longest_length = len(noun_phrase)
                noun_phrase_temp = noun_phrase
            if (str(noun_phrase_temp) in
str(noun_phrase)) or (str(noun_phrase) in
str(noun_phrase_temp)) and (noun_phrase_temp != None):
                if len(noun_phrase) > longest_length:
                    longest_lenght = len(noun_phrase)
                    noun_phrase_temp = noun_phrase
            elif (str(noun_phrase_temp) not in
str(noun_phrase)) and (str(noun_phrase) not in
str(noun_phrase_temp)) and (noun_phrase_temp != None):

longest_noun_phrases.append(noun_phrase_temp)
                noun_phrase_temp = noun_phrase
                longest_length = len(noun_phrase)
        longest_noun_phrases.append(noun_phrase_temp)
        return longest_noun_phrases
```

```
1 test_4 = text_triplets(sentences)
2 for sentence in sentences:
3     sentence = nlp(sentence)
4     nph = test_4.find_noun_phrases(sentence)
5     new_nph = test_4.longer_noun_phrase(nph)
6     print(new_nph)
```

```
[Маша, Саша, трубе]
[Мой друг Иван, городе Москва]
[Мама, новый магазин]
[Мама, овощей, фруктов, ягод]
[машине, музыка]
```

Рисунок 6 – Результат выполнения функций выделения именных форм

2.3.5 Составление триплетов из полученных именных и глагольных конструкций

На данном этапе, этапе формирования триплетов, для каждого предложения имеется набор именных групп, и набор глагольных групп, которые требуется объединить, для получения триплетов вида «концепт-связь-концепт», который будет отражать главную суть предложения, где концептами будут являться именные группы, а связью между ними – глагольная группа.

Данный этап выполняется в три фазы: сначала мы формируем триплеты путем сравнения токенов слов, таким образом определяя положение именной группы, будет ли она являться субъектом предложения и стоять слева от глагольной конструкции, или являться объектом и находится справа. Для сравнения токенов используется метод сравнения значений токенов из библиотеки textacy [7]. Затем мы собираем массив, которые будет состоять из трех элементов: первым элементов будет набор всех субъектов в предложении, вторым – глагольная группа, а третьим – набор всех объектов в предложении. И последней фазой будет распаковка все субъектов и объектов, так, чтобы каждому одному субъекту соответствовал один объект, и связью являлась одна глагольная группа. Графическая реализация работы алгоритма поиска и формирования триплетов изображена в приложении А.4

Ниже представлены функции, выполняющие формирование всех триплетов в каждом предложении. Результат выполнения функций представлен на рисунках 7 и 8.

```
def find_triplet(self, sentence):
        doc = nlp(sentence)
        verb_phrases = self.get_verb_phrases(doc)
        noun_phrases = self.find_noun_phrases(doc)
        verb_phrase = None
        noun_phrase = None
        if (len(verb_phrases) > 1):
```

```
            verb_phrase =
self.longer_verb_phrase(list(verb_phrases))
        else:
            verb_phrase = verb_phrases[0]
        if (len(noun_phrases) > 2):
            noun_phrase =
self.longer_noun_phrase(noun_phrases)
        else:
            noun_phrase = noun_phrases
        if len(noun_phrase) == 2:
            left_noun_phrase = noun_phrase[0]
            right_noun_phrase = noun_phrase[1]
        else:
            words = str(sentence).split()
            left_noun_phrase =[]
            right_noun_phrase = []
            verb_pos = None
            i = 0
            for j in range(len(words)):
                if words[j] in str(verb_phrase) and
len(words[j]) > 4:
                    verb_pos = i
                if words[j] in str(noun_phrase) and
verb_pos == None:

left_noun_phrase.append(noun_phrase[i])
                    i += 1
                else:
                    right_noun_phrase =
noun_phrase[i:len(noun_phrase)]
```

```
        return (left_noun_phrase, verb_phrase,
right_noun_phrase)


    def triplets_array(self):
        for sentence in self.sentences:
            (a,b,c) = self.find_triplet(sentence)
            self.edges.append([a,c])
            self.verbp.append(b)
        c = 0
        for i in self.edges:
            if type(i[0]) == list:
                if type(i[1]) == list:
                    for a in i[0]:
                        for b in i[1]:
                            self.edges_2.append([a,b])

self.verbp_2.append(self.verbp[c])
                else:
                    for a in i[0]:
                        self.edges_2.append([a,i[1]])

self.verbp_2.append(self.verbp[c])
            else:
                if type(i[1]) == list:
                    for b in i[1]:
                        self.edges_2.append([i[0],b])

self.verbp_2.append(self.verbp[c])
                else:
```

```
                    self.edges_2.append([i[0],i[1]])
                    self.verbp_2.append(self.verbp[c])
            c += 1
```

```
test_5 = text_triplets(sentences)
for sentence in sentences:
    (a,b,c) = test_5.find_triplet(sentence)
    print(a, "\t", b, "\t", c)
```

```
[Маша, Саша]     сидели на       [трубе]
Мой друг Иван    живет в         городе Москва
Мама     пошла в         новый магазин
[Мама]   купила          [овощей, фруктов, ягод]
машине   громко играет   музыка
```

Рисунок 7 – Результат выполнения функции формирования триплетов

```
test_6 = text_triplets(sentences)
test_6.triplets_array()
edges = test_6.edges_2
labels = test_6.verbp_2
print(edges)
print(labels)
```

```
[[Маша, трубе], [Саша, трубе], [Мой друг Иван, городе Москва], [Мама, новый магазин], [Мама, овощей], [Мама, фруктов], [Мама, я
год], [машине, музыка]]
[сидели на, сидели на, живет в, пошла в, купила, купила, купила, громко играет]
```

Рисунок 8 – Результат выполнения функции развертывание триплетов

2.3.6 Лемматизация триплетов

После формирования всех триплетов для каждого предложения, нужно провести лемматизацию каждой словоформы. Во многих предложениях одна и таже словоформа может находится в разных видах, и при формировании концептуальной карты они будут являться разными концептами. Чтобы избежать подобной ошибки, мы находим лемму – начальную форму каждого слова.

Лемматизация – процесс приведения всех изменяемых форм слова к единому значению. Снижает вариативность одного и того же слова, что повышает качество анализа текста.

Алгоритм лемматизации основан на поиске наиболее подходящего варианта слова по словарю. При анализе текстовой информации обычно используются данные, полученные после процесса токенизации,

подразумевающего разделение текста на отдельные слова или предложения. После сопоставления со словарем все словоформы одного слова заменяются на одно конкретное значение.

В языках со сложным словообразованием (например, русском) может потребоваться помимо стандартных словарей использовать дополнительные, учитывающие специфику речи. Отдельно к процессу лемматизации подключатся словари сленга, аббревиатуры и сокращений.

Ниже представлена функция, выполняющая лемматизацию. Результат выполнения функции представлен на рисунке 9.

```
def lemmatize(self):
        for i in range(len(self.edges_2)):
            for j in range(len(self.edges_2[i])):
                self.edges_2[i][j] =
self.edges_2[i][j].lemma_
            self.verbp_2[i] = self.verbp_2[i].lemma_
```

```
test_7 = text_triplets(sentences)
test_7.triplets_array()
test_7.lemmatize()
edges = test_7.edges_2
labes = test_7.verbp_2
print(edges)
print(labels)
```

```
[['маша', 'труба'], ['саша', 'труба'], ['мой друг иван', 'город москва'], ['мама', 'новый магазин'], ['мама', 'овощ'], ['мама',
'фрукт'], ['мама', 'ягода'], ['машина', 'музыка']]
[сидели на, сидели на, живет в, пошла в, купила, купила, купила, громко играет]
```

Рисунок 9 – Результат выполнения функции лемматизации триплетов

2.3.7 Визуализация предложений на основе dependency меток

На этапе разбиения на предложения, при токенизации предложений SpaCy автоматически проводит Dependency parsing над предложениями. Dependency parsing – процесс, который анализирует грамматическую структуру предложения и выявляет связи слов, а так же типы связей между ними. Так же связи в предложениях можно представить в виде графа зависимостей, с помощью встроенного визуализатора зависимостей – displaCy. В основе методе

Dependency parsing лежит принцип грамматики зависимостей, которые предполагает, что предложения можно представить в виде графа зависимости, где слова соединены между собой дугами, и каждая метка дуги является типом связи между двумя словами. Корнем предложения всегда является глагол. Далее от глагола задаются вопросы для выявления зависимостей с другими словами. От присоединенных к корню предложения сущностей тоже возможно задать вопрос, чтобы получить, например, характеристику объекта. Данный метод очень хорошо подходит для языков, у которых свободный порядок слов.

Ниже представлена функция, выполняющая визуализацию предложений в тексте в виде графов зависимостей. Результат выполнения функции представлен на рисунке 10.

```
def displacy_show(self):
        for sentence in self.sentences:
            sentence = nlp(sentence)
            displacy.render(sentence, style='dep',
jupyter=True)
```

```
1 test_8 = text_triplets(sentences)
2 test_8.displacy_show()
```

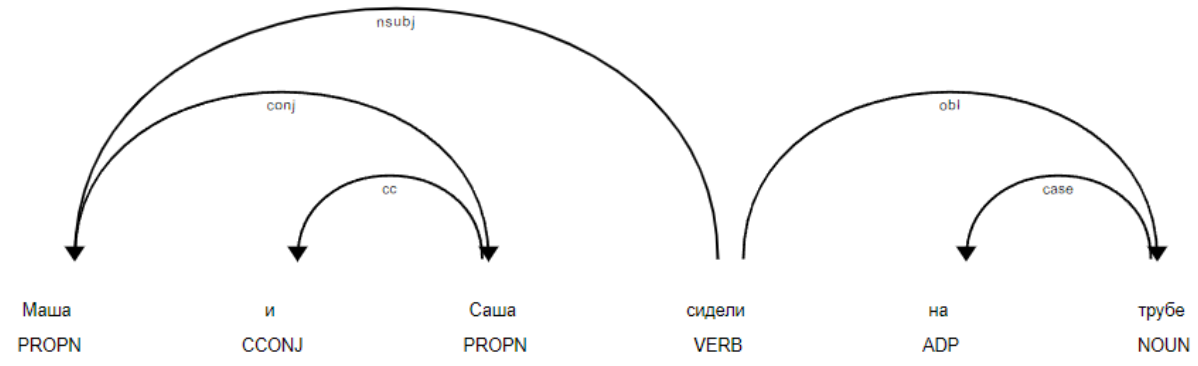


Рисунок 10 – Результат выполнения функции визуализации предложений в виде графов зависимости на одном из предложений в исходном тексте

2.3.8 Составление и визуализация концептуальной карты

Последний этап – это составление, на основе полученного набора триплетов, словаря, где ключ – это составной элемент, состоящий из двух вершин – концептов, а значение ключа – связь между ними. При помощи библиотеки network – простой библиотеки для визуализации графов, происходит визуализация концептуальной карты, которая является выдержкой из текста.

Ниже представлена функция, выполняющая подготовку данных и последующую визуализацию графа. Результат выполнения функций представлен на рисунке 11.

```
  def edge_labels_create(self):
        for i in range(len(self.edges)):
            self.edge_labels[tuple(self.edges[i])] =
self.labels[i]


    def visualize(self):
        edges = self.edges
        G = nx.DiGraph()
        G.add_edges_from(edges)
        pos = nx.spring_layout(G, k=.6)
        plt.figure(figsize=(17,15))
        nx.draw(
            G, pos, edge_color='black', width=1,
linewidths=1,
            node_size=4300,
node_color='pink',node_shape="o", alpha=1,arrows=True,
            labels={node: node for node in G.nodes()}
        )
        nx.draw_networkx_edge_labels(
```

```
        G, pos,
        edge_labels=self.edge_labels,
        font_color='red'
    )
    plt.axis('off')
    plt.show()
```

```
call_3 = visualizer(edges, labels)
call_3.edge_labels_create()
call_3.visualize()
```

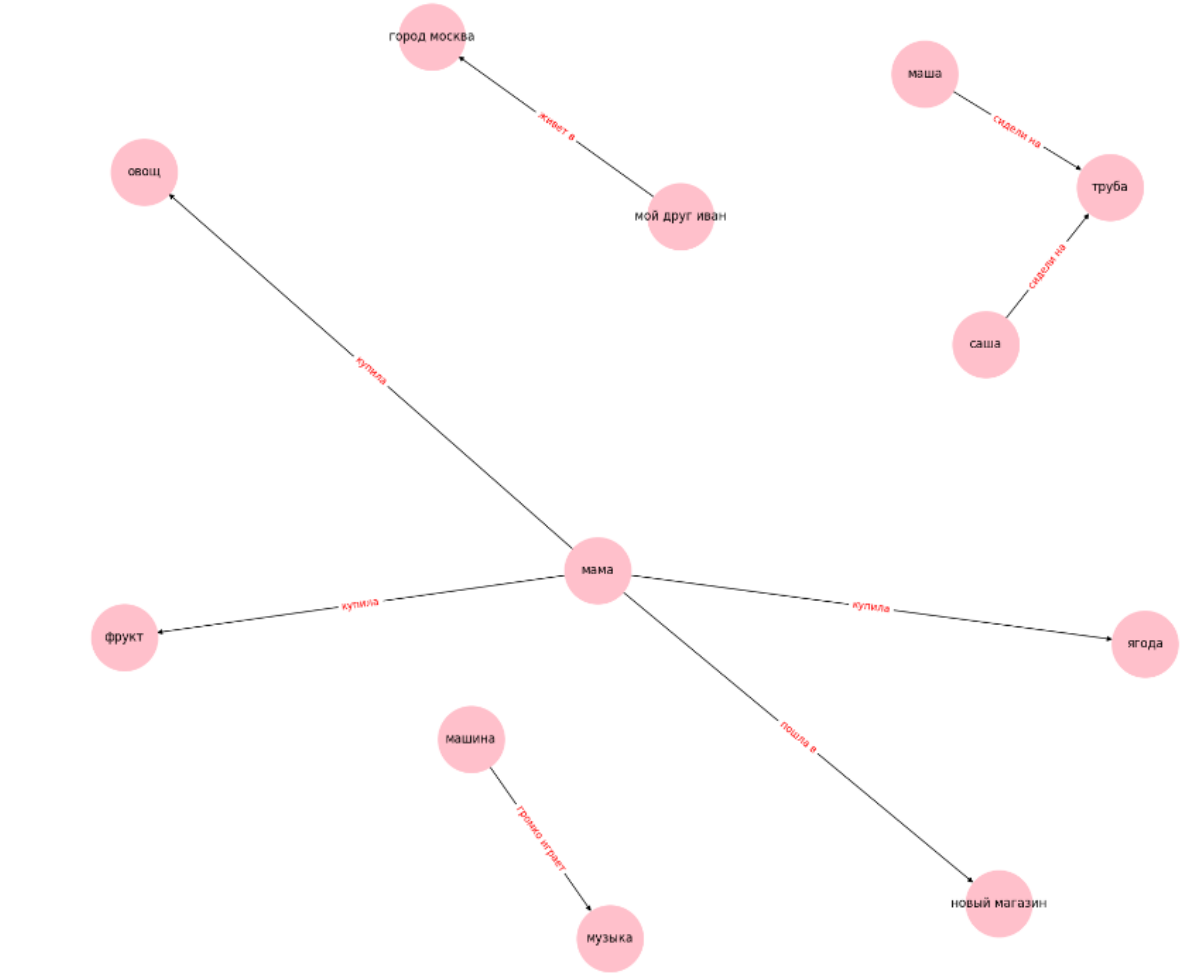


Рисунок 11 – Результат выполнения функций визуализации концептуальной карты

2.3.9 Оценка полученной концептуальной карты

Оценка полученной концептуальной карты очень сложная задача, так как требует нахождение четырех экспертов в области теории концептуальных карт, которые составят свои варианты концептуальных карт на исходном тексте, проведут оценку карт друг друга, тем самым будет высчитана метрика inter human-annotator agreement, а так же проверят и оценят полученную автоматизированным путем концептуальную карту, тем самым возможно будет посчитать метрику machine-human agreement. В данный момент в области теории концептуальных карт очень мало экспертов, составление карт которых можно было бы принять за «Золотой стандарт».

3 ИССЛЕДОВАТЕЛЬСКАЯ ЧАСТЬ

3.1 Метод автоматической генерации концептуальных карт Concept map mining

Семантическая сеть представляет собой структуру для представления знаний в виде взаимосвязанных вершин и дуг [8]. Концептуальная карты является особым типом семантической сети, гибкой и ориентированной на человека. Она представлена в виде направленного графа, где узлы представляют концепты, а дуги – связи между ними [9].

Понятие концептуальной карты приписывается к учению теоретика Джозефа Новака, где его группа исследователей описала процесс обучения человека, как пожизненный процесс усвоения новых концептов и связей между ними в личную концептуальную структуру [1]. Новак адаптировал модель семантической сети и создал понятие концептуальной карты, как инструмент для графического представления концептуального понимания информации обучающимся в определенной области. Его изначальной идеей было, что концептуальная карта должны быть нарисована рукой обучающегося после определения всех главных идей и их классификации в иерархической манере. Топология концептуальных крат может принимать разные формы от иерархической до иерархической и формами, управляемые данными.

В общем виде, иерархическая концептуальная карта может быть определена следующим набором

$$CM = \{C, R, T\} \quad (1)$$

где

$C = \{c_1, c_2, \dots, c_n\}$ – набор концептов. Каждый концепт $c_i \in C; 1 \leq i \leq n$ это слово или словосочетание, и оно является уникальным в C.

$R = \{r_1, r_2, \dots, r_m\}$ является набором связей между концептами. Каждая связь $r_i \in R = \left(c_p, c_q, l_j\right); p \neq q; 1 \leq p \leq n; 1 \leq q \leq; 1 \leq j \leq m,$ соединяет два концепта $c_p, c_q \in C$. Метка l_i определяет тип связи r_j.

$T = \{t_1, t_2, \dots, t_s\}; t_{k-1} < t_k < t_{k+1}; 1 < k < s$ является отсортированным набором иерархических уровней в концептуальной карте. Каждый уровень соответствует набору концептов, которые делят один и тот же уровень иерархии в концептуальной карте.

Concept map mining представляет собой процесс извлечения информации из одного или более документов для автоматического создания концептуальной карты. Созданная карта является обобщенной сводной исходного текста [10].

С точки зрения CMM документ может быть представлен как набор

$$D = \{C_d, R_d\} \quad (2)$$

где

$C_d = \{c_{\mathrm{d1}}, c_{\mathrm{d2}}, \dots, c_{dn}\}$ – набор всех концептов

$R_d = \{r_{d1}, r_{d2}, \dots, r_{dm}\}$ – набор всех связей между концептами, которые могут быть выделены из текста. Упрощенный алгоритм работы метода Concept map mining представлен в приложении А.2.

Тремя основными фазами Concept map mining являются: извлечение концептов, извлечение связей, обобщение. Первый этап является определением и извлечением всех концептов, которые представлены субъектами и объектами в тексте – обычно это существительные и именные фразы [11]. Когда известна синтаксическая и семантическая связь между концептами, то возможно извлечь связь между ними, что является целью второго этапа. Последним этапом является обобщение документа, где на выходе мы получим набор иерархически выстроенных триплетов концепт-связь-концепт. Концептуальная карта предназначена для анализа человеком, поэтому желательна чтобы она не имела слишком большого количества концептов. Так же важно, чтобы в концептуальной карте использовались термины, которые использовались в изначальном документе.

Первые наброски метода CMM были представлены в работе Трочима, которые использовал синтаксический анализ для решения данной задачи. Группа людей во время собрания накидывала некоторое количество утверждений, которое относилось к теме собрания. Каждый участник оценивал все утверждения, создавая индивидуальную матрицу. После все матрицы суммировались в групповой аппроксимирующий массив. Наиболее важные утверждения выбирались с метода многомерного шкалирования. Такой подход и сейчас используется в CMM [12].

3.2 Подходы использующиеся в Concept map mining

Процесс Concept map mining может быть выполнен методами обработки естественного языка, такие как извлечение информации (IE – Information Extraction), информационный поиск (IR – Information Retrieval) и автоматическое суммирование (Automatic summarization). IE – это задача автоматического извлечения структурированных данных, таких как сущности и связи из неструктурированных или слабоструктурированных машиночитаемых документов. IR представляет собой процесс поиска неструктурированной информации, удовлетворяющий информационным потребностям, а автоматическое суммирование является процессом сокращения набора данных для создания подмножества, содержащего наиболее важную информацию в исходном содержании [13]. Результатом автоматического суммирования могут быть выдержка или реферат. Выдержка является подмножество текста, содержащего наиболее важную информацию, выделенную из оригинала без изменений, в то время как рефератом является перефразированная выдержка из текста [14]. Методы, использующиеся в этих областях, являются статистическими методами на основе правил и методами машинного обучения. Относительно недавно появился интерес к объединению конечных автоматов с моделями условной вероятности, как например модели энтропии Маркова и условно случайные поля [13]. Большинство классических методов

суммирования также являются числовыми и основаны на модели взвешивания, такими, как например term frequency-inverse document frequency (TF-IDF). Методы машинного обучения часто обеспечивают точное извлечение на основе классификации, использующей бинарную или нечеткую логику. Такие методы могут быть использованы как основной метод или в гибридных системах для обеспечения ресурсами других процессов. Современные подходы включают гибридные подходы с использованием алгоритмов в комбинации со сторонними наборами данных [14], [15], суммированием, основанным на нечеткой логике и роевом интеллекте [16]. В области обработки естественного языка, численные методы могут быть обогащены словарями терминов [13]. Но существует проблема со словарями: они должны быть заранее созданы для определенной области. Так же ограничивающий фактор использования лингвистических методов – это отсутствие нужных методов и инструментов для многих языков.

3.3 Анализ подходов к реализации

На этапах выделения концептов и на этапе выделения связей между объектами и субъектами требуется провести разбиение текста на предложения, токенизацию предложений, а также синтаксический анализ предложения. Синтаксический анализ как входное данное получает результат морфологического парсера, строит граф зависимостей на основе меток зависимостей полученных при dependency parsing, которое будет отражать структуру исходного предложения. Каждая связь, которая связывает пару словоформ, является подчинительной связью между ними, направление связи указывает на направление данной подчинительной связи.

В основном синтаксическая структура представлена двумя вариантами: грамматикой зависимости, или грамматикой непосредственно-составляющих. В грамматике зависимостей предложение представляется как состоящее из слов и синтаксических отношений между ними. Выделяется главное слово (сказуемое или подлежащее), от него проводится стрелка к зависимым членам

предложения, далее к зависимым членам предложения второго ранга, и т.д. Предложение представляется как совокупность слов и синтаксических связей между ними, а их порядок роли почти не играет. Такое представление еще называют «графами Теньера».

Грамматика непосредственно-составляющих оперирует единицами, которые называются непосредственно-составляющие. Непосредственно-составляющая – каждая из двух конструкций максимального объема, которое можно выделить в составе каждого предложение, а также в составе каждого следующего непосредственного-составляющего.

Грамматика непосредственно-составляющих отличается от других теорий способом представления структуры предложения. Если каждая из других грамматик оперирует одной, одинаковой по формату единицей, то грамматика непосредственно-составляющих за единицу принимает последовательности словоформ, обладающие разными характеристиками, а также последующие словоформы входят в предыдущие.

К тому же в грамматике зависимостей синтаксическая структура предложения почти не отражает его линейную структуру, а именно линейный порядок слов в предложении. Чтобы это исправить требуется дополнить синтаксическую структуру предложения дополнительным указанием на его линейную структуру.

Но и у грамматики непосредственно-составляющих есть свои трудности:

- Когда принцип линейного соседства противоречит с синтаксическими связями в предложении.
- Когда нету четкий оснований для выбора способа членения по непосредственно-составляющим, тогда в таком случае предложение может иметь несколько синтаксических деревьев

В нынешних реализациях синтаксический анализ на основе грамматики зависимостей имеет большую популярность, чем на основе грамматики непосредственно-составляющей. Возможная причина такого решения скорее всего сводится к тому, что грамматика зависимостей лучше подходит для

анализа языков с гибкой структурой порядка словоформ, и где лексические и морфологические признаки имеют больше веса, нежели чем порядок слов в предложении. На рисунке 12 представлен граф зависимости для предложения «программа решает сложную задачу», а на рисунке 13 представлен пример разбора того же самого предложения, но уже на основе грамматики непосредственно-составляющих.

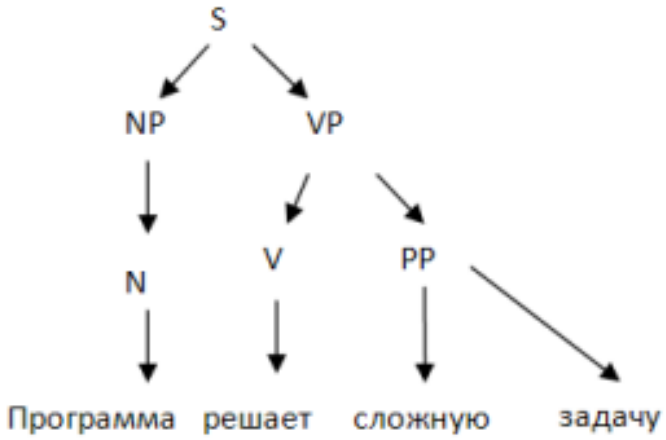


Рисунок 12 – Пример разбора предложения на основе грамматики непосредственно-составляющих

Здесь, на рисунке 12, N – существительное, V – глагол, PP – предложная группа, NP – именная группа, VP – глагольная группа, S – предложение. Изначально все предложение считается непосредственно-составляющей, а затем делится на именную и глагольную группу, которые в свою очередь делятся на более простые составляющие.

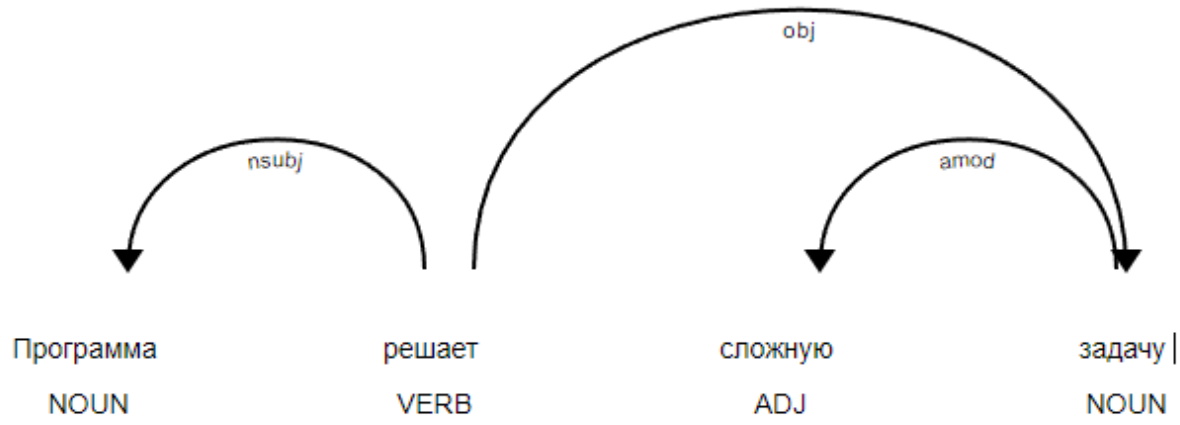


Рисунок 13 – Пример разбора предложения на основе грамматики зависимостей

Здесь же, на рисунке 13, продемонстрирован пример разбора того же самого предложения, но уже основываясь на другой грамматике – грамматике зависимостей. Метки у стрелок означают тип зависимости, а метки у словоформ – результат частеречной разметки: NOUN – существительное, VERB – глагол, ADJ – прилагательное, amod – адъективный модификатор, nsubj – является номинальным субъектом и является протоагентом клаузы, obj – номинальный объект.

Для точной работы парсеров, которые основываются на правилах, требуется большой набор лингвистических шаблонов. Эти шаблоны создаются людьми, имеющими большую компетенцию в данной области – лингвистами.

Помимо представления в виде графа зависимости, синтаксический анализ можно представить в виде таблице, где первый столбец будет являться текстом токена, второй – частью речи данного токена, затем видом связи токенов между собой, и наконец, является ли словоформа корнем предложения. Пример разбора того самого предложения можно увидеть в таблице 3.

Таблица 3. Пример синтаксического анализа предложения.

Токен	Часть речи	Метка связи	Является ли словоформа корнем
Программа	NOUN	nsubj	False
решает	VERB	root	True
сложную	ADJ	amod	False
задачу	NOUN	obj	False

3.4 Анализ существующих решений

Разработчики по всему миру занимаются решением задач обработки естественного языка (NLP), и в основном используют английский язык. Из-за этого некоторые модели для обработки текста лучше подходят для определенных языков, и хуже себя показывают для других. Различия в структуре и синтаксисе естественных языков создают сложности в создании унифицированных средств обработки языка.

Идея Universal Dependencies [6] была предложена для упрощения работы над мультиязычными машинным переводом и обеспечения совместимости синтаксических деревьев. Благодаря этому проекту была достигнута лингвистическая согласованность аннотаций и появились возможности для расширения языковых моделей. Universal Dependencies включает боле 100 корпусов для 70 языков, включая русский, приведенных к одному формату.

С увеличением спроса на обработку текстовых данных появляется множество программных средств для обработки естественного языка, которые предлагают решения как для типичных задач, таких как POS-tagging, NER, IR, IE, лемматизация и других, так и для конкретных нестандартных проблем пользователей. Рассмотрим ряд широко известных решений.

SpaCy [17]: это библиотека для обработки естественного языка, которая обладает специальными средствами для построения систем анализа текста. Она

предлагает свои нейросетевые модели для синтаксического анализа, классификации и извлечения именованных сущностей, а также обеспечивает высокую производительность. SpaCy реализована на языка Python/Cython для удобства разработки и эффективной работы.

Slovnet [18] – это открытая Python-библиотека, интегрированная с другими проектами Natasha, такими как Nerus, Razdel, Navec, Yargy. Каждая решает свою собственную задачу: Razdel – токенизация, и сегментация текста на предложения, Navec – компактные эмбеддинги для русского языка, Yargy – извлечение фактов, основанное на правилах и Nerus – корпус данных аннотированный POS тегами, NER тегами. Она проста в установке и не требует жестких зависимостей с другими модулями. Slovnet обучается на новостной модели, но переобучение и до обучение является сложной задачей из-за ограниченного доступа к скриптам и сложной структуре модели.

Stanza [19]: это библиотека от StanfordNLP, разработанная университетом Стэнфорда. Она предоставляет широкий набор инструментов для обработки языка на различных языках.

NLTK [20]: это пакет библиотек и программ для символьной и статистической обработки естественного языка на языке Python. NLTK широко используется в исследованиях NLP и разработке алгоритмов машинного обучения. Однако синтаксический анализ в NLTK основан на грамматике составляющий, что вызывает множество ошибок при попытке анализа текстов на русском языке.

StanfordNLP [21]: это модуль для Python, который используется как интерфейс для работы с клиентом CoreNLP на Java. Скорость работы данного парсера ниже по сравнению с другими решениями, так как требуется установленная Java на компьютере.

DeepPavlov [22]: Открытый фреймворк и библиотека глубокого обучения, предназначенные для разработки и применения моделей обработки естественного языка (NLP) и диалоговых систем. Он разрабатывается командой специалистов из Networks and Deep Learning лабаратории компании «OpenAI

Russia» и ряда университетов. DeepPavlov предоставляет широкий спектр инструментов и моделей для решения различных задач в области обработки NLP, включая машинный перевод классификацию текста, извлечение информации, ответы на вопросы, чат-боты и многое другое. Он основан на использовании глубоких нейронных сетей и предоставляет готовые модели, которые можно использовать «из коробки», а также гибкую архитектуру для создания собственных моделей и компонентов. DeepPavlov поддерживает несколько языков программирования, включая Python, и предлагает удобный интерфейс для разработки, обучения и использования моделей.

Проведем сравнительный анализ этих решений по критериям, которые представлены в таблицах 4-12.

Обозначим каждое решение своей переменной, которые отображены в таблице 4.

Таблица 4. Обозначения вариантов парсеров.

Название решений	**Обозначение**
SpaCy	X1
Slovnet	X2
Stanza	X3
NLTK	X4
StanfordNLP	X5
DeepPavlov	X6

Таблица 5. Критерии сравнения решений.

Критерий	**Обозначение**
Легкость установки	K1

Продолжение Таблицы 5.

Критерий	**Обозначение**
Возможность токенизации	K2
Легкость использования	K3
Вид парсера звисимости	K4
Возможность визуализации	K5
Возможность переобучения моделей	K6

Таблица 6. Вербально-числовая шкала для K1(Легкость установки)

Легко	**Требуются доп. усилия**	**Сложно**
1	0.5	0

Таблица 7. Вербально-числовая шкала для K2 (Возможность токенизации)

Из коробки	**Требуются установка доп. модулей**	**невозможна**
1	0.5	0

Таблица 8. Вербально-числовая шкала для K3(Легкость использования)

Легко	**Средне**	**Сложно**
1	0.5	0

Таблица 9. Вербально-числовая шкала для К4(Вид парсера зависимости)

Основанный на грамматике зависимостей	**Основан на грамматике непосредственно-составляющих**
1	0

Таблица 10. Вербально-числовая шкала для К5(Возможность визуализации)

Есть возможность	**Есть возможность используя доп. модули**	**Нет возможности**
1	0,5	0

Таблица 11. Вербально-числовая шкала для К6(Возможность переобучения моделей)

Есть возможность и имеются все средства для этого	**Есть возможность, но дополнительных сведений нету**	**Нет возможности переобучения моделей**
1	0.5	0

Таблица 12. Нормированные значения сравниваемых решений

Код фактора	**Весовой коэффициент**	**Значение показателей**					
		X1	**X2**	**X3**	**X1**	**X5**	**X6**
К1	0,156	1	1	0,5	0,5	0	1
К2	0,251	1	0,5	1	1	1	1
К3	0,178	1	1	0,5	1	0	1
К4	0,235	1	1	1	0	1	1

Продолжение таблицы 12.

Код фактора	**Весовой коэффициент**	**Значение показателей**					
		X1	**X2**	**X3**	**X4**	**X5**	**X6**
K5	0,195	1	0	0,5	0	0,5	1
K6	0,275	1	0	0,5	0,5	0,5	1
$Y_{j=\sum_i^n \alpha_i k_{ij}}$		1,29	0,695	0,888	0,645	0,721	1,29

Ранжирование вариантов по предпочтительности:

$X1 = \text{X6} \succ X3 \succ X5 \succ X2 \succ X4$

В результате применения метода взвешенной суммы нормированных показателей сравнения можно сделать вывод, что варианты X1 (SpaCy) и X6 (DeepPavlov) являются наиболее предпочтительными вариантами. При тестировании DeepPavlov было выявлено, что модели работают некорректно для требуемых задач, а именно: задачи извлечения концептов и задачи извлечения связей. В задаче извлечения концептов, концептами являются только известные личности, мелькавшие в новостях, причем не всегда корректно происходит и их нахождение, так же некоторые города, страны, но остальные сущности в виде название предметов, и простых объектов не поддаются распознанию, и для них даже не было объявлено меток. Так же наблюдалась проблема с унификацией меток зависимостей, частеречных меток: от задачи к задаче некоторые метки пропадали, и не поддавались распознаванию. Большинство проблем можно было бы решить переобучением модели на собственном корпусе данных, но при попытке обучения, вылезало огромное количество ошибок, что делало обучение модели невозможным.

Таким образом самым предпочтительным вариантом остается библиотека SpaCy, которая обладает всеми нужными инструментами для решения большинства поставленных задач, так же в преимущества данной библиотеки можно добавить наличие отлично оберточной библиотеки Textacy, которая

помогает решить оставшиеся задачи, которые невозможно было бы решить при помощи чистой библиотеки SpaCy.

3.5 Обучение модели SpaCy

Синтаксические модели составляют прогнозы на основе данных, которые были использованы в процессе обучения. Обычно точно таких моделей можно повысить, до обучив модель на дополнительных примерах, которые являются специфичными для конкретной задачи пользователя. Дополнительное обучение моделей, особенно в контексте синтаксического анализа, может быть очень полезным.

Модели обычно включают в себя следующие компоненты:

- Двоичные веса для частеречной разметки, парсера зависимостей и задачи распознавания именованных сущностей.
- Лексические записи в словаре, включающие слова и их независимые атрибуты, такие как форма или написание.
- Файлы данных, такие как правила лемматизации и таблицы поиска.
- Векторы слов, представляющие собой многомерные смысловые представления слов и позволяющие провести сравнительный анализ слов
- Параметры конфигурации, такие как язык, а также параметры обработки, которые настраивают SpaCy при загрузке модели.

Для обучения модели SpaCy можно добавлять дополнительные параметры прямо в интерфейсе. Сами примеры должны состоять из текстовых данных и соответствующего количества списка меток, на основе которых будет проводится обучение модели.

Рисунок процесса обучения модели представлен на рисунке 3. Начинается он с тренировочных данных, которые используются для до обучения модели. Текст в тренировочных данных должен соответствовать данным, с которыми модель будет работать в дальнейшем, чтобы достичь лучше результатов.

Графическая реализация поэтапного обучения модели SpaCy представлена в приложении А.5 [23].

После подготовки тренировочных данных можно приступить к самому процессу обучения. Обычно требуется множество итераций обучения, чтобы эффективно оптимизировать веса параметром модели. В SpaCy для этого используется метод стохастического градиентного спуска. Чтобы получить надежные градиенты, примеры перемешиваются для каждой итерации и передаются модели батчами, а не по одному. Это позволяет улучшить оценку градиента.

Для обучения парсера необходимо иметь размеченный корпус текстов на русском языке, где для каждого предложения есть дерево разбора и указаны части речи для всех его членов предложения.

В одной строке файла хранятся атрибуты одного члена предложения:

- Порядковый номер;
- Токен;
- Обработанный токен;
- Метка зависимости;
- POS метка;
- NER метка;
- Граммемы слова;
- Родитель в графе зависимости;
- Тип зависимости;

3.6 Оценка точности созданной концептуальной карты

Оценка автоматически созданной концептуальной карты является очень сложной, которая работает со знаниями и значит, что она очень субъективна. В данный момент нету ни одной точной метрики для оценки точности созданной концептуальной карты. Мною предложено проводить оценку с использованием концептуальных карт экспертов, которые будут приняты как «Золотой

Стандарт», который используется в данный момент для автоматической оценки эссе – тоже очень субъективной и комплексной задачи [24]. Алгоритм оценки представляется следующим образом: случайно выбирается набор документов/текстов в документе из всего набора, по которым от двух до четырех экспертов создадут концептуальные карты. Далее будет посчитана human inter-annotator agreement метрика, так же будет посчитана machine-human agreement метрика. После подсчета проведется сравнение двух показателей и если показатель human—machine agreement больше или равен human inter-annotator agreement, то производительность системы допустима.

Human inter-annotator agreement – является метрикой уровня согласия или согласованности между человеческими оценщиками при выполнении конкретной задачи. Цель расчета заключается в оценке надежности человеческий аннотаций и определении согласованности мнений различных оценщиков. Оно помогает понять субъективный характер задачи аннотации и предоставляет информацию о качестве и достоверности аннотированных данных.

Machine-human agreement – метрика согласования между человеком и машиной, также известная как корреляция между машиной и человеком, является метрикой которая оценивает согласованность аннотации между сформированных с помощью машинных алгоритмов и человеческими оценками или аннотациями.

ЗАКЛЮЧЕНИЕ

При разработке модуля формирования концептуальных карт из текста для русского языка были получены следующий результаты:

- Была исследована предметная область обработки естественного языка.
- Была исследована предметная область формирования концептуальных карт.
- Были сформулированы функциональные требования к модулю.
- Был проведен сравнительный анализ существующих решений и выбран наиболее оптимальный для выполнения поставленных задач.
- Выбрана оптимальная модель машинного обучения.
- Разработан модуль формирования концептуальных карт из текста для русского языка.
- Успешно проведены тесты разработанного модуля.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1) Novak J.D., Cañas A.J., The theory underlying concept maps and how to construct and use them //IHMC CmapTools 2006-01, Rev 01-2008, Florida Institute for Human and Machine Cognition, Pensacola Fl, Tech. Rep., January 2008.

2) Плунгян В.А. Классификация морфологический значений //Общая морфология: Введение в проблематику: Учебное пособие. – 2-е, исправление. - М.: Едиториал УРСС, 2003. – С. 107. – 384 с. – 2000 экз. – ISBN 5-354-00314-8.

3) Trained pipelines for Russian SpaCy [Электронный ресурс] – URL: https://spacy.io/models/ru (Дата обращений 02.06.2023).

4) Pymorphy2 [Электронный ресурс] – URL: https://pymorphy2.readthedocs.io/en/stable/ (Дата обращения 03.06.2023).

5) OpenCorpora [Электронный ресурс] - URL: https://github.com/OpenCorpora/opencorpora (Дата обращения 03.06.2023).

6) Universal Dependecies [Электронный ресурс] – URL: https://universaldependencies.org/ru/index.html (Дата обращения 04.06.2023).

7) Textacy: Documentation [Электронный ресурс] – URL: https://textacy.readthedocs.io/en/latest/ (Дата обращения 01.06.2023).

8) Kramer B.M., Mylopoulos J. Representation, knowledge, Encyclopedia of Artificial Intelligence //Wiley-Interscience Publication, 1987, vol. 2, pp. 206–214.

9) McNeese M.D., Zaff B.S., Peio K.J., Snyder D.E., Duncan J.C., McFarren M.R. An advanced knowledge and design acquisition methodology for the pilot’s associate //Harry C. Armstrong Aerospace Medical Research Laboratory, Human Systems Division, Tech. Rep., 1990.

10) Villalon J.J., Calvo R.A., Concept map mining: A definition and a framework for its evaluation //Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on, vol. 3, pp. 357–360, 2008.

11) Villalon Jorge, Calvo Rafael Analysis of a gold standard for concept map mining-how humans summarize text using concept maps //URL:

https://cmc.ihmc.us/cmc2010papers/cmc2010- b4.pdf (дата обращения: 09.04.2023).

12) Trochim W.M.K. An introduction to concept mapping for planning and evaluation, Evaluation and Program Planning //vol. 12, no. 1, pp. 1–16, January 1989.

13) Manning C.D., Schütze H. Foundations of Statistical Natural Language Processing //Cambridge University Press, 1999.

14) Das D., Matrins A.F.T. A survey on automatic text summarization // URL:
http://www.cs.cmu.edu/~afm/Home_files/Das_Martins_survey_summarization.pdf (дата обращения 11.04.2023)/

15) Spärck Jones K., Automatic summarising: The state of the art //Inf. Process. Manage., vol. 43, pp. 1449–1481, November 2007.

16) Binwahlan M.S., Salim N., Suanmali L. Fuzzy swarm-based text summarization //Journal of Computer Science, vol. 5, no. 5, pp. 338–346, 2009

17) SpaCy: API Documantation [Электронный ресурс] – URL: https://spacy.io/api (Дата обращения 03.06.2023).

18) Slovnet – Python library for deep-learning NLP modeling for Russian language [Электронный ресурс] – URL: https://github.com/natasha/slovnet (Дата обращения 04.06.2023).

19) Stanza: Documentation [Электронный ресурс] – URL: https://stanfordnlp.github.io/stanza/index.html (Дата обращения 05.06.2023).

20) NLTK: Documentation [Электронный ресурс] – URL: https://stanfordnlp.github.io/stanza/index.html (Дата обращения 05.06.2023).

21) StanfordNLP: CoreNLP overview [Электронный ресурс] – URL: https://stanfordnlp.github.io/stanza/index.html (Дата обращения 05.06.2023).

22) DeepPavlov: Documentation [Электронный ресурс] – URL: https://docs.deeppavlov.ai/en/master/ (Дата обращения 05.06.2023).

23) Training Pipelines & models [Электронный ресурс] – URL: https://spacy.io/usage/training (дата обращения 06.06.2023).

24) Hearst M.A. The debate on automated essay grading //IEEE Intelligent Systems and their Applications, vol. 15, no. 5, pp. 22-37, Sept.-Oct. 2000, doi: 10.1109/5254.889104.

ПРИЛОЖЕНИЕ А

ГРАФИЧЕСКИЕ МАТЕРИАЛЫ

А.1 – Алгоритм работы модуля формирования концептуальных карт из текста для русского языка

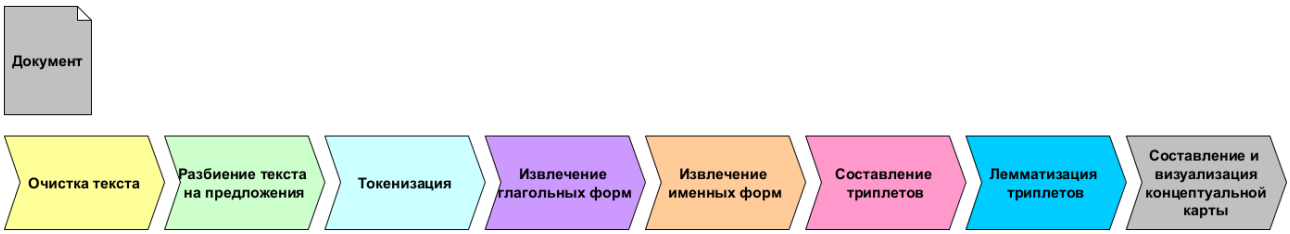
А.2 – Алгоритм работы метода Concept Map Mining

А.3 – Алгоритм работы извлечения галгольных сущностей

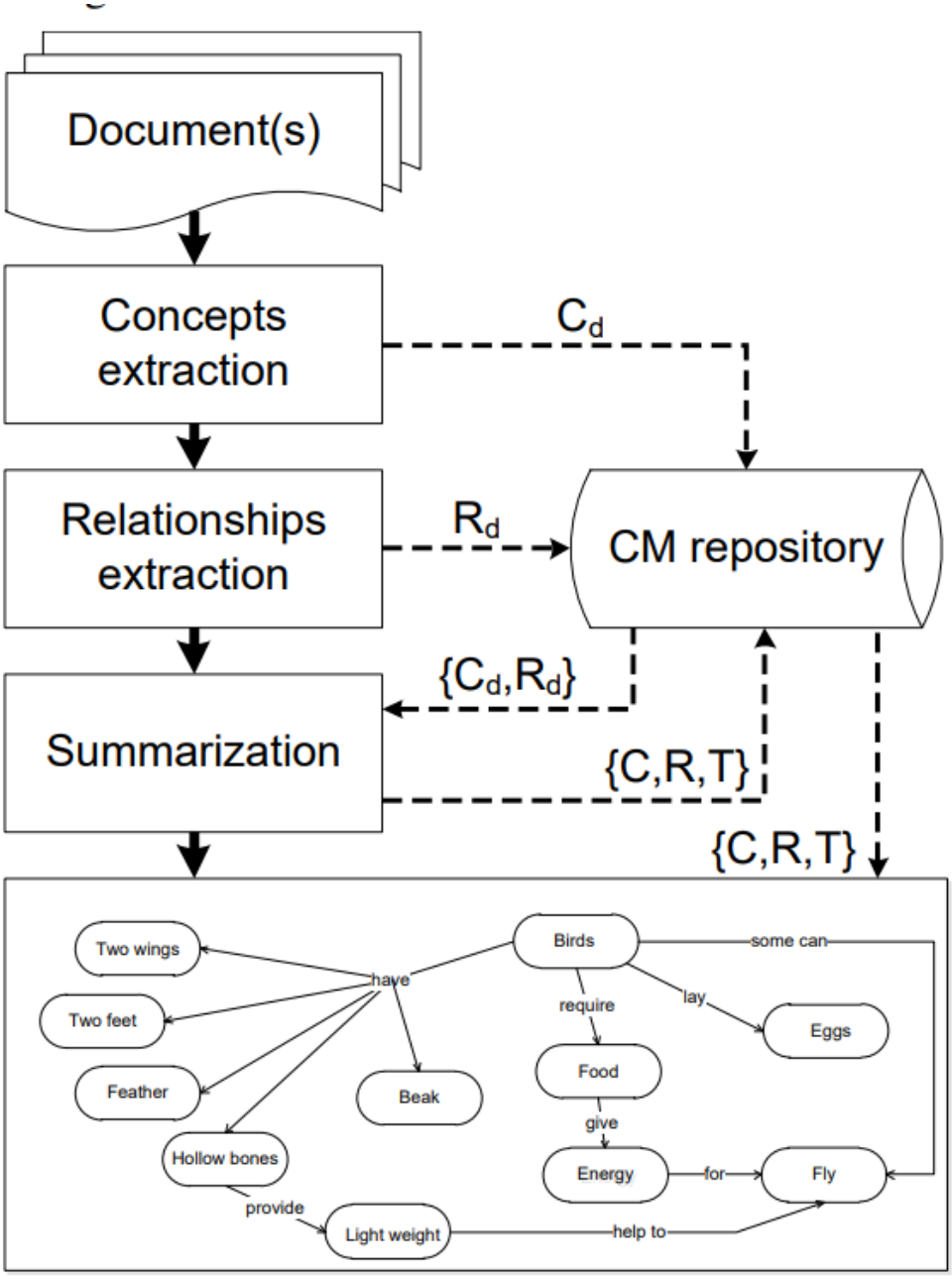
А.4 – Алгоритм работы извлечения триплетов

А.5 – Обучение модели SpaCy

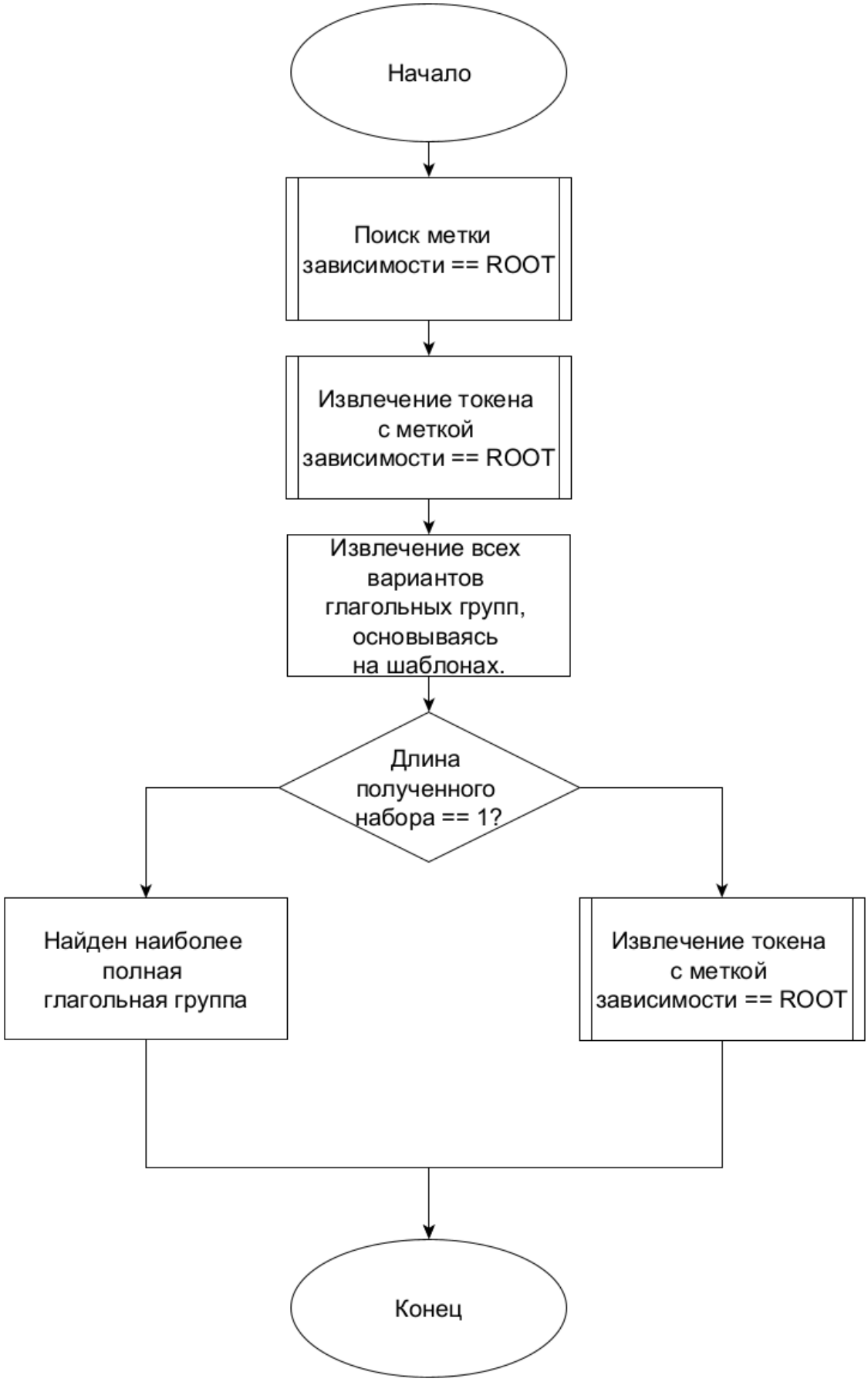
А.1 – Алгоритм работы модуля формирования концептуальных карт из текста для русского языка



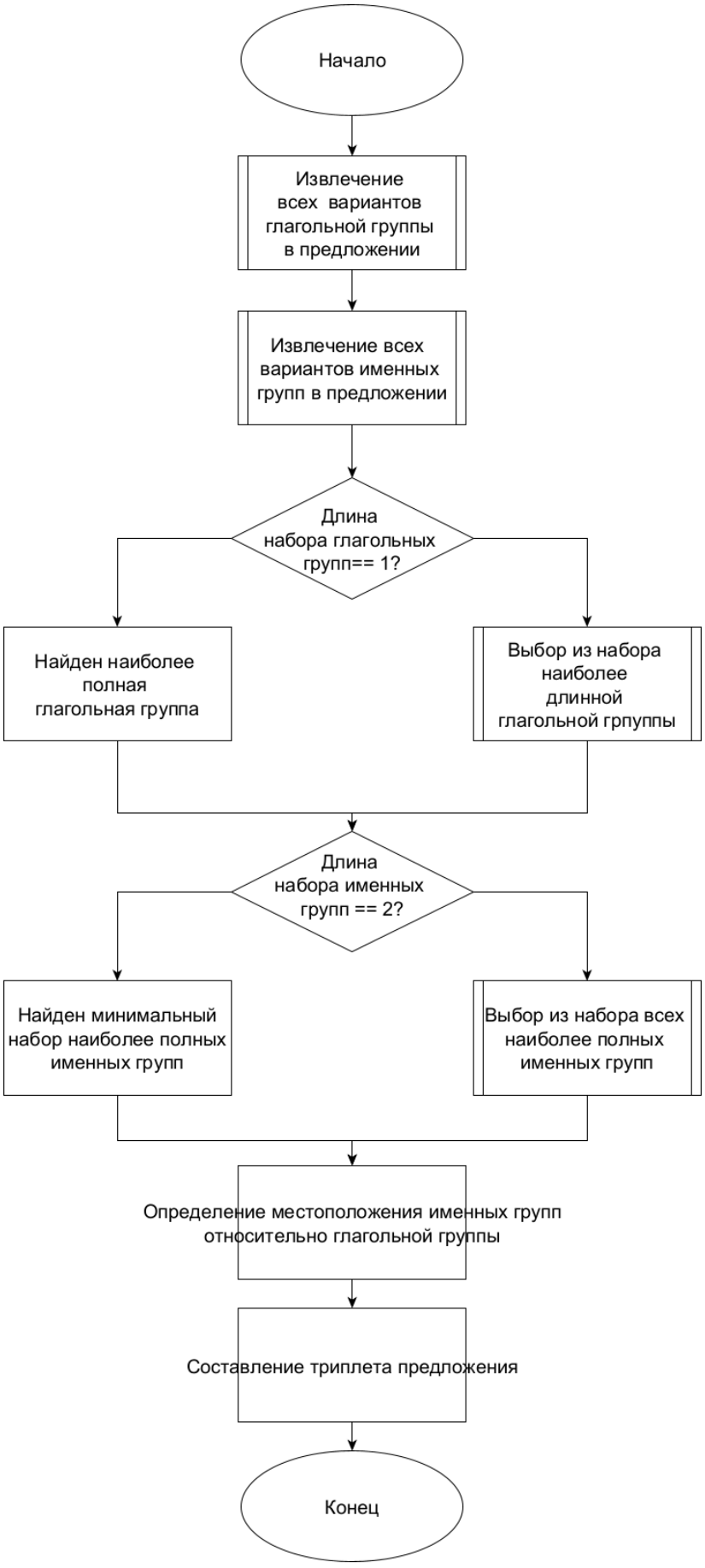
А.2 – Алгоритм работы метода Concept Map Mining



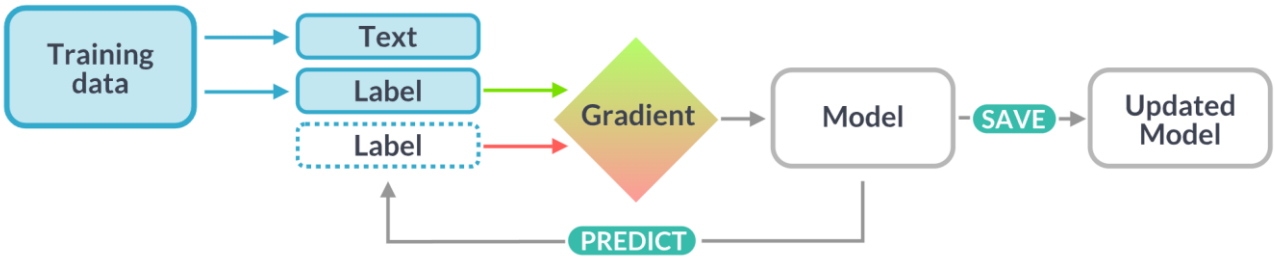
A.3 – Алгоритм работы извлечения галгольных сущностей



А.4 – Алгоритм работы извлечения триплетов



A.5 – Обучение модели SpaCy


Training data
Text
Label
Label
Gradient
Model
SAVE
Updated Model
PREDICT

ПРИЛОЖЕНИЕ Б

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Кафедра «Системы обработки информации и управления»

«Утверждаю»
Научный руководитель
_____Максаков А.А.
«___» _____ 2023 г.

Модуль формирования концептуальных карт на основе анализа текста для русского языка

Техническое задание
(вид документа)

писчая бумага
(вид носителя)

7
(количество листов)

ИСПОЛНИТЕЛЬ:

_____ Евсеев Георгий Андреевич
"_" _____ 2023 г.

Москва – 2023

1. Наименование

Модуль формирования концептуальных карт на основе анализа текста для русского языка.

2. Основание для разработки

Основанием для разработки является задание на выпускную квалификационную работу, подписанное руководителем выпускной работы и утвержденное заведующим кафедрой ИУ5 МГТУ им. Н.Э. Баумана.

3. Исполнитель

Студент четвёртого курса группы РТ5-81Б Евсеев Г.А.

4. Цель работы

Назначением работы является создание алгоритма генерации концептуальных карт, основанного на морфологическом и синтаксическом анализе текста на русском языке.

5. Содержание работы

5.1 Задачи

5.1.1. Исследовать предметную область, определить функциональные задачи.

5.1.2. Разработать архитектуру модуля формирования концептуальных карта на основе анализа текста на русском языке.

5.1.3. Реализовать обработку входного текста.

5.1.4. Реализовать разбиение текста на предложения.

5.1.5. Реализовать метод токенизации предложений.

5.1.6. Реализовать метод POS-tagging’а предложений.

5.1.7. Реализовать метод Dependency parser’а предложений.

5.1.8. Реализовать метод визуализации Dependency меток предложений.

5.1.9. Реализовать метод выделения именных конструкций из предложений.

5.1.10. Реализовать метод выделения глагольных конструкций из предложений.

5.1.11. Реализовать метод выделения триплетов вида “концепт-отношение-концепт” из предложений.

5.1.12. Реализовать метод лемматизации полученных триплетов вида “концепт-отношение-концепт”.

5.1.13. Реализовать метод визуализации концептуальной карты.

5.2 Требования к функциональным характеристикам

Разрабатываемая система должна выполнять следующие функции:

5.2.1. Обработка входного текста.

5.2.2. Разбиение текста на предложения.

5.2.3. Выделение глагольных конструкций из предложений.

5.2.4. Выделение именных конструкций из предложений.

5.2.5. Составление триплетов вида “концепт-отношение-концепт” для каждого предложения.

5.2.6. Лемматизация триплетов вида “концепт-отношение-концепт”.

5.2.7. Визуализация предложений на основе dependency меток слов.

5.2.8. Составление и визуализация концептуальной карты, на основе полученных триплетов.

5.3 Требования к входным и выходным данным

5.3.1 Требования к входным данным

- Данные, полученные от пользователя: - текст.

5.3.2 Требования к выходным данным

Выходные данные представляют собой визуализацию концептуальной карты, на основе полученного текста, путем его синтаксического и морфологического анализа.

5.4 Требования к надежности

Система должна надежно и устойчиво функционировать, при возникновении ошибки должно выводиться сообщение об ошибке.

5.5 Требования к архитектуре программного изделия

Модуль формирования концептуальных карт на основе анализа текста для русского языка должен быть реализован на Python;

5.6 Требования к составу программных средств

Для работы клиентской части требуется наличие:

1. Python 3.9+;
2. Библиотека Spacy;
3. Библиотека Textacy;
4. Библиотека Re;
5. Библиотека Networkx;

5.7 Требования к составу технических средств

Минимальные системные требования для работы клиентской части:

1. Процессор с тактовой частотой 1200 ГГц;
2. Оперативная память - 16 ГБ;
3. Видеокарта - GeForce GTX 1070 или выше
4. Видеоадаптеры и мониторы, способные обеспечить графический режим 1024*768 точек
5. Жёсткий диск объемом 15 ГБ;
6. Манипулятор «мышь» или другое указывающее устройство;
7. Клавиатура.

6. Этапы работы

График выполнения отдельных этапов работ приведен в соответствии с приказом об организации учебного процесса в 2022/2023 учебном году.

Таблица 1: Этапы разработки

1	Наименование этапа и содержание работ	Сроки исполнения
1.	Утверждение темы ВКРБ и научного руководителя	Декабрь 2022 г.
2.	Представление документов “Задание на выполнение ВКРБ”, “Календарный план выполнения ВКРБ” (Смотр № 1)	Декабрь 2022 г.
3.	Формулирование проблемы, цели и задач работы	Январь 2023 г.
4.	Разработка первой части РПЗ “Постановка задач разработки”	Февраль 2023 г.
5.	Разработка технического заданий	Март 2023 г.
6.	Представление рабочих материалов технического заданий (Смотр № 2)	Март 2023 г.
7.	Разработка второй части РПЗ “Исследовательская часть”	Март 2023 г.
8.	Разработка программы методики испытания	Апрель 2023 г.
9.	Разработка третьей части РПЗ “Конструкторско-технологическая часть”	Апрель 2023 г.
10.	Защита макета программы	Май 2023 г.
11.	Разработка заключения, приложений, оформления работы	Май 2023 г.
12.	Подготовка доклада и презентации	Май 2023 г.
13.	Получения заключения научного руководителя	Май 2023 г.
14.	Допуск работы к защите на ГЭК (нормоконтроль)	Май 2023 г.
15.	Защита работы на ГЭК	Июнь 2023 г.

7. Техническая документация

По окончании работы предъявляется следующая техническая документация:

1. Техническое задание;
2. Рабочий материал по выполняемому проекту;
3. Программа и методика испытаний;

4. Графический материал по проекту в формате презентации.

8. Порядок приема работы

Прием и контроль программного изделия осуществляется в соответствие с методикой испытаний (см. документ «Программа и методика испытаний»).

9. Дополнительные условия

Данное техническое задание может уточняться в установленном порядке.

ПРИЛОЖЕНИЕ В

ПРОГРАММА И МЕТОДИКА ИСПЫТАНИЙ

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Утверждаю

_____.

«___» _____2023 г.

Согласовано

_____

«___» _____2023 г.

Программа и методика испытаний выпускной квалификационной работы бакалавра «Модуль формирования концептуальных карт на основе анализа текста для русского языка»

Программа и методика испытаний

(вид документа)

писчая бумага

(вид носителя)

6

(количество листов)

Исполнитель:

студент группы РТ5-81

Евсеев Г.А.

_____

«___» _____2023 г.

Москва, 2023

Содержание

1 Объект испытаний

Объектом испытаний является модуль формирования концептуальных карт на основе анализа текста на русском языке.

Система предназначена для синтаксической и морфологической обработки текста, формировании и визуализации концептуальной карты, основанной на полученном тексте.

2 Цель испытаний

Испытание проводится с целью проверки соответствия программного изделия требованиям, описанным в техническом задании.

3 Состав предъявляемой документации

При проведении испытаний предъявляются следующие документы:

1. техническое задание выпускной квалификационной работы бакалавра;
2. программа и методика испытаний выпускной квалификационной работы бакалавра.

4 Технические требования

Минимальные системные требования для работы клиентской части:

1. Процессор с тактовой частотой 1200 ГГц;
2. Оперативная память - 16 ГБ;
3. Видеокарта - GeForce GTX 1070 или выше
4. Видеоадаптеры и мониторы, способные обеспечить графический режим 1024*768 точек
5. Жёсткий диск объемом 15 ГБ;

6. Манипулятор «мышь» или другое указывающее устройство;

5 Порядок проведения испытаний

Испытания системы будут проводиться в следующем порядке.

1. Запуск программного изделия.
2. Тестирование базовых операций.
3. Закрытие программного изделия.

Приемочные испытания включают проверку:

1. Полноты и качества реализации функций, указанных в ТЗ.
2. Полноты действий, доступных пользователю, и их достаточность для функционирования системы.
3. Реакции системы на ошибки пользователя.

6 Последовательность проведения испытаний

Последовательность проведения испытательных мероприятий приведен в Таблице 1.

Таблица 1 - Последовательность проведения испытаний

№	Действие	Ожидаемый результат	№ п. ТЗ
1	Запуск ячейки обработки входного текста.	Вернуть очищенный текст	5.2.1
2	Запуск ячейки разбиения текста на предложения.	Вернуть массив, содержащий все предложения, введенного текста	5.2.2
3	Запуск ячейки выделения глагольных конструкций из полученных предложений.	Вернуть массивы содержащие все глагольные конструкции, которые были выделены из всех предложений в тексте.	5.2.3

4	Запуск ячейки выделения именных конструкций из полученных предложений.	Вернуть массивы содержащие все именные конструкции, которые были выделены из всех предложений в тексте.	5.2.4
5	Запуск ячейки составления на основе выделенных именных и глагольных конструкций триплеты вида “концепт-отношение-концепт”, которые отражают главную суть каждого предложения в тексте.	Вернуть массивы содержащие в все триплеты вида “концепт-отношение-концепт”, которые отражают главную суть каждого предложения в тексте.	5.2.5
6	Запуск ячейки лемматизации триплетов “концепт-отношение-концепт”.	Вернуть массив лемматезированных концептов, и массив лемматизированных отношений.	5.2.6
7	Запуск ячейки визуализации предложений на основе dependency меток слов.	Вернуть графы предложений, основанных на dependency метках каждого слова в этих предложениях.	5.2.7
8	Запуск ячейки составления и визуализации концептуальной карты, на основе полученных триплетов.	Вернуть визуализированную концептуальную карту текста.	5.2.8

7 Результат испытаний

Испытание считается пройденным успешно, если в процессе демонстрации:

1. Программа была успешно запущена.
2. Все выводы и промежуточные значения соответствуют ожиданию.