Практическая работа №1 «Морфологический анализ»

1 **Цель работы**: освоение практических навыков морфологического анализа текста

2 **Порядок выполнения работы:**

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.

3 **Краткие теоретические сведения**

3.1 Морфологический анализ

Морфология – это уникальный раздел филологической науки, русского языка, призванный изучить разделы речи и используемые в ней слова с качественной и количественной точки зрения. Морфологический анализ – это уникальный шанс и способ разобрать текст «дословно». С его помощью автор сможет определить следующие моменты:

– правильность написания слов – проверка орфографии и грамматики в тексте способствует повышению качества информации;
– правильность и уместность употребления слов в тексте/предложении и пр. В данном ракурсе морфологический анализ позволяет понять владение терминологией, навыки по ее грамотному и эффективному использования, возможности автора в части самовыражения;
– наполненность текста определенными конструкциями и словами, необходимость перестроений и переформулировок с целью «разгрузки» и придания материалу «легкости» и ясности.

Фактически морфологический анализ способствует тотальному погружению в написанный или изученный материал, более качественной оценке и определению сути исследования.

В целом морфологический анализ предполагает дословный и описательный разбор конкретной части текста. Он призван расчленить исследование на более мелкие составляющие и определить суть, роль каждого элемента в нем.

На практике выделяют два вида морфологического анализа:

Дословный, когда исследователь изучает каждое слово с точки зрения морфологии, разбирая его принадлежность к конкретной группе, способ использования (падеж, склонение и пр.), правильность написания и значение (в котором оно употреблено). Именно данная схема применяется чаще всего школьниками и студентами первых курсов, осваивающих филологический или лингвистический профиль;

Варианты проведения морфологического анализа:

Попредложный. В данном случае автор изучает состав предложения с выделением задействованных слов, их форм, принадлежности, а также предоставляет характеристику конструкции в целом: ее характер, вид, стиль и пр. Этот формат применим в редакторском, корректорском деле. Также им успешно пользуются критики, филологи и лингвисты при оценке сторонних трудов на качество.

Дословный морфологический анализ призван установить значение использованного термина, правильность и уместность его употребления/написания, роль и пр. «Попредложная схема» позволяет оценить качественный и количественный состав текста: частота использования сложных и простых конструкций, какие слова чаще применимы в предложениях, какова роль составного элемента в абзаце/тексте в целом, грамотность автора, способы упрощения (повышение удобочитаемости) текста и пр.

Рисунок 1 – Этапы реализации морфологического анализа

Для проведения морфологического анализа исследователю важно разобраться в простейших элементах и правилах русского языка, изучить составные части. Общая (стандартная) методика проведения данного вида «мыслительно-разъяснительной работы» выглядит следующим образом:

– Изучаем текст;

– Делим на составные части для анализа: дословный или попредложный;

– Определяем существенные моменты в отношении объекта исследования: подробно описываем его принадлежность к части речи, определяем начальную форму, стилистику, «индивидуальность» (особенности употребления) и пр.

– Определяем общую характеристику и правильность употребления объекта в материале.

– Алгоритм проведения морфологического анализа может зависеть непосредственно от объекта исследования. Дословная метода различается в зависимости от изучаемого элемента.

Дословная схема проведения морфологического анализа различается в зависимости от исследуемой части речи. Сейчас мы приведем краткий алгоритм реализации данного приема для каждой из них:

Имя существительное: начальная форма – отношение к лексико-грамматическому разряду (имя собственное или нарицательное с уточнением) – одушевленное/неодушевленное – род — число – падеж – склонение – синтаксическая функция;

Имя прилагательное: начальная форма (приводим задействованное слово в единственное число, мужской род, именительный падеж, ответив на простой вопрос: «Какой?») – лексико-грамматический разряд (качественное, относительное или притяжательное) – полная/краткая форма – род, число и падеж + связь между сочетаемым существительным/местоимением/числительным – тип склонения – роль в предложении;

Местоимение: начальная форма – разряд по значению (в какой группе относится: личное, возвратное, вопросительное и пр.) – соотношение с другими частями речи – род, число и падеж – синтаксическая роль;

Рисунок 2 – Дословный морфологический анализ

Дословный морфологический анализ

Глагол: инфинитив – основы и класс анализируемого слова – спряжение – вид – переходный/непереходный – возвратный/невозвратный – наклонение – время – лицо – род – синтаксическое место в тексте;

Причастие: начальная форма – вид – залог – время – для страдательной формы важно определение рода, числа и падежа – образовательный суффикс и основа (от какого слова) – синтаксическая роль;

Деепричастие: вид – возвратное/невозвратное – основа+суффикс – место и сущность в тексте;

Наречие: часть речи – разряд по значению – группа внутри разряда – синтаксический акцент (роль);

Предлог и союз: часть речи – разряд по строению и значению – связь с другими словами в тексте, миссия;

Частица, модальное слово, междометие: разряд по значению и строению, структуре.

Алгоритм проведения морфологического анализа в отношении предложения в целом выглядит следующим образом:

Оценка простой конструкции: тип по цели высказывания (повествовательное, вопросительное, побудительное и пр.) – эмоциональная окраска (вопросительное/восклицательное) – структура (одно- или двусоставное) – оценка второстепенных членов (распространенное/нераспространенное) – осложняющие компоненты (чем осложнено) – оценка структуры (полное/неполное);

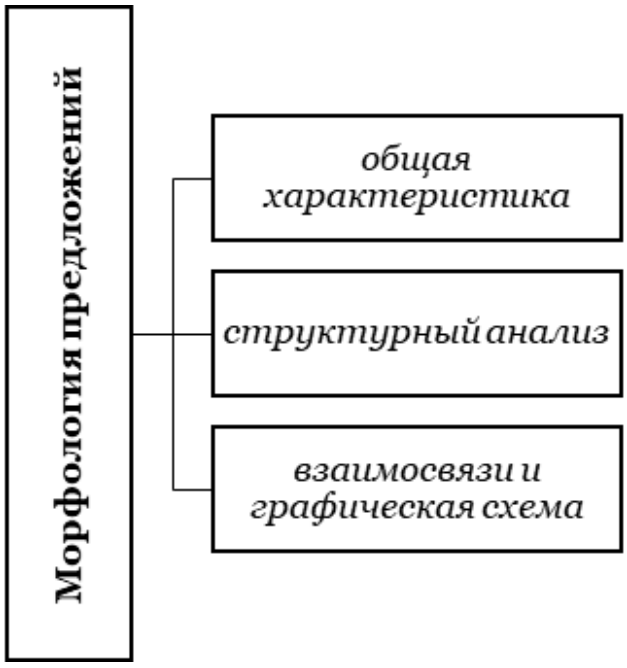

Рисунок 3 – Алгоритм проведения морфологического анализа текста

Алгоритм проведения морфологического анализа текста

Оценка сложных конструкций: конкретизация типа предложения по цели высказывания – определение типа по эмоциональной окраске – констатация вида (сложносочиненное или сложноподчиненное) – выделение частей и их количеств (открытая/закрытая, расчлененная/нерасчлененная) – определение

средств связи между частями предложения (союзы, порядок частей речи, вопрос от одной части к другой и пр.) – характер смысловых взаимосвязей – графическая схема предложения.

На сегодняшний день реализация морфологического анализа возможна в двух вариациях. Первый – ручной, когда автору или исследователю, квалифицированному специалисту предстоит ознакомиться с материалов, провести вычитку и самостоятельно проанализировать составные части на основе имеющихся знаний, опыта и пр. В данном случае обработка большого объема информации отнимет немало времени и сил, потребует тотальной концентрации и точности, владения пером и языком.

Второй способ – автоматизированный, который стал доступен каждому благодаря современным технологиям и продвижению НТП. В настоящее время существует свыше нескольких сотен программ и приложений, специальных сервисов, готовых обработать информацию, провести морфологический и иной анализ за считанные минуты, облегчая миссию эксперта или автора.

Прежде чем провести морфологический анализ, необходимо выделить из текста отдельные слова. В связи с этим вместе с системой морфологического анализа иногда поставляется подсистема графематического анализа. Входной поток символов разбивается на токены нескольких классов: буквенные последовательности, числа, цифробуквенные комплексы, пунктуация, разделители, иероглифы. При этом каждый класс токенов имеет собственный набор тегов, в частности, для слов это может быть язык (кириллица или латиница) и регистр.

Анализ несловарных слов является важной функцией систем морфологического анализа. Как показывает практика, составить абсолютно полный словарь практически невозможно, ведь естественный язык является постоянно развивающейся системой. При увеличении размера анализируемых текстов до нескольких десятков миллиардов слов словарь так и не выходит на насыщение. Это связано с появлением текстов и лексики из новых предметных областей; всё новыми ошибками, появляющимися в текстах; новыми именами собственными, которые авторы придумывают для беллетристики, или которые приходят с новостями из заграницы. На одном из круглых столов конференции Диалог2 коллеги из Яндекса говорили, что пользователи делают около 480 опечаток в день в слове одноклассники. В месяц количество уникальных опечаток превышает 1500. При этом от 80% до 90% всех опечаток отстоят от оригинала на одну ошибку.

Ещё одним ярким примером активного словообразования является немецкий язык с его составными существительными. В немецком языке принято некоторые понятия, которые должны описываться несколькими словами, давать одним словом. Это новое слово получается за счёт «склеивания» входящих в него слов по определенным правилам. В качестве примера слово Donaudampfschifffahrtskapit¨an — капитан рейса, выполняемого пароходом по Дунаю: Donau — Дунай, Dampfschiff — пароход (в свою очередь тоже составное слово), Fahrt — рейс, Kapit¨an — капитан. Там же говорится о

том, что река Дунай вполне может быть заменена на Неву или другую реку, в результате чего будет получено новое слово. Подобное образование слов особенно принято в формальных областях человеческой деятельности: названия мероприятий, министерств, регламентов и др.

Аналогичная ситуация наблюдается и в русском языке, хотя и в значительно меньшей степени. Несколько неудачно, но вполне художественно смотрятся такие фразы, как, например, краснопогонное общество или интернетоговорящая публика. Аналогично образуются слова через дефис: серо-буро-малиновый, кубик-кубик и др. Помимо этого, не следует скидывать со счетов слова новояза, принятые в некоторых сообществах: даунлоадить, бэксайд, гуфи, запитонить, напитонил и в продакшн и пр.

В связи с наличием подобных явлений в языке в систему морфологического анализа необходимо вводить модуль анализа несловарных слов. Обычно он реализуется с помощью набора эвристик, таких как отсечение префиксов, аналогия по окончанию и правила для слов с дефисом.

Ещё одной важной функцией является снятие морфологической омонимии. В разных системах реализуются два различных похода для решения этой задачи: контекстное и бесконтекстное снятие. Бесконтекстное снятие выполняется на основе подсчёта статистики по размеченному корпусу, а контекстное снятие — с помощью классификатора, настроенного при помощи одного из методов машинного обучения.

Мы можем рассчитать, например, сколько раз встретилась каждая из лексем. Для этого надо предварительно провести лемматизацию. В итоге для нашего примера мы получим вектор, показанный в таблице 1.

Таблица 1: Вектор лексем для текста «Косой косой косил косой косой за песчаной косой»

Токен	За	Коса	Косить	Косой (прил)	Косой (сущ)	песчаный
Частота	1	2	1	2	1	1

Аналогичным образом можно построить вектор частот для любого текста, состоящего из произвольного количества предложений. Вектор может быть построен как для лемм, так и для словоформ или токенов, в зависимости от того, какая перед нами стоит задача. Векторное представление позволяет перейти от текста к его описанию в некотором пространстве. Представим себе, что каждая лемма задаёт собственное направление в некотором многомерном пространстве (размерность которого будет равна количеству лемм в тексте). В таком случае, текст можно представить как точку или вектор в этом многомерном пространстве. Более того, если у нас имеется несколько текстов, то мы можем объединить все леммы (словоформы, токены) этих текстов и получить пространство большей размерности. В этом новом пространстве

можно будет представить в виде точки или вектора каждый из имеющихся текстов.

Переход к многомерному пространству позволяет измерять расстояния между текстами, то есть степень в которой они похожи или не похожи друг на друга. Логично предположить, что если в двух текстах употребляются примерно одни и те же слова, то эти тексты посвящены одной (или сходной) теме. И наоборот, если в тексте отсутствуют одинаковые слова, есть большая вероятность, что они относятся к разным предметным областям (если только над текстом не поработал опытный копирайтер, поставивший себе задачу не оставить в переписываемом тексте ни одного слова).

Заметим, что привычная Евклидова метрика работает в таком пространстве очень и очень плохо. Но в данной главе мы не будем останавливаться на вопросе определения меры сходства между текстами. Рассмотрим пока особенности хранения таких векторов. Средняя новостная заметка содержит порядка сотни словоупотреблений (или нескольких сотен в случае аналитической статьи), среди которых встречается в несколько раз меньше лемм. Однако если мы будем анализировать все новостные заметки, полученные из одного источника, размер текста будет уже около сотни миллионов словоупотреблений и порядка 300 000 лемм[3]. Но при этом мы продолжаем работать с отдельными заметками или их группами. Получается, что для работы с одной заметкой в сотню слов нам необходим вектор размерностью в несколько сотен тысяч параметров. При количестве заметок, приближающимся к миллиону, разместить всю информацию в оперативной памяти становится проблематично. А если нам необходимо работать с сотнями новостных лент?

Для того, чтобы избежать подобной траты памяти мы можем сократить объем хранимой информации. Для слов, которые не встретились в данной заметке, соответствующее значение в векторе будет равно нулю.

Если слова будут храниться в алфавитном порядке, то мы можем хранить только те слова, частота которых не равна нулю. В этом случае можно предположить, что все слова, которые расположены между двумя словами в векторе, встретились ноль раз. То есть, фактически мы храним только ненулевые значения, а размерность вектора может быть приведена к общему количеству слов во всех текстах.

Обратите внимание, что в этом примере слово «стали» может быть разобрано и как глагол («они стали лучше справляться»), и как существительное («кислородно-конверторный способ получения стали»). На основе одной лишь информации о том, как слово пишется, понять, какой разбор правильный, нельзя, поэтому анализатор может возвращать несколько вариантов разбора.

В примере со словом «косой», мы увидели, что одному и тому же токену может соответствовать несколько разных словоформ. Подобное явление называется лексической неоднозначностью. Лексическая неоднозначность включает в себя несколько явлений. Омонимия — это явление, при котором два

слова сходны по написанию и звучанию, но различные по смыслу. Различают лексическую (совпадение форм разныхлексем), грамматическую (совпадение токенов одной лексемы), синтаксическую (различные корректные трактовки одной и той же последовательности слов) и другие виды омонимии. В нашем случае песчаная коса,косая коса и косой косой в соответствующих формах будут именно омонимами. Лексическая неоднозначность, помимо омонимии, включает в себя также различные варианты анализа слова в рамках одной лексемы: токен косой может быть проанализирован одновременно как именительный падеж мужского рода и творительный падеж женского рода в рамках одной лексемы ⟨косой, прилагательное⟩.

Снятие (или разрешение) омонимии — этап анализа текста, на котором проводится выбор единственного варианта морфологического анализа для каждого токена. Если словарь не содержит данный токен, будем называть этот токен несловарным. Если с данным токеном связано более одной словоформы, буем называть такой токен неоднозначным. Как уже было сказано выше, словоформа содержит в себе лемму, часть речи и набор грамматических параметров. Следовательно, неоднозначность может проявить себя в одной из этих частей или их комбинации.

Помимо несловарным и однозначных слов введём для токенов ещё четыре класса омонимии.

Неоднозначные по параметрам — в анализе присутствуют словоформы с различными множествами грамматических параметров, но совпадающей леммой и частью речи. Например, прилагательное косой может выражать как именительный падеж мужского рода, так и творительный падеж женского рода.

Неоднозначные по части речи — в анализе присутствуют словоформы, совпадающие по лемме, но отличающиеся по части речи. Так как части речи не совпадают, то наборы параметров у словоформ также будут отличаться. В связи с этим сравнение параметров проводиться не может. Например, прилагательное и существительное раненый.

Неоднозначные по лемме — в анализе присутствуют словоформы, отличающиеся по лемме, но имеющие одинаковую часть речи. Здесь параметры могут как совпадать, так и отличаться. Примером совпадающих параметров является токен смели — третье лицо единственного числа от лемм сметь и смолоть (ударение у токенов будет различаться, но мы не увидим этого в тексте). Примером различающихся параметров будет являться токен вина, соответствующий именительному падежу единственного числа леммы вина и именительному падежу множественного числа леммы вино.

Неоднозначные по части речи и лемме — в анализе присутствуют словоформы, отличающиеся как по лемме, так и по части речи. Сравнение параметров здесь также проводиться не может. Примером здесь может служить токен стекло, соответствующий именительному падежу единственного числа существительного стекло и третьему лицу среднего рода прошедшего времени глагола стечь.

В зависимости от постановки задачи, морфологический анализатор может возвращать разную информацию. Если нам необходимо рассчитать вектор частотности употребления слов в тексте, это может быть только лемма или лемма и часть речи. Для языков с невысокой флективностью применяется стемминг — определение основы слова путём отбрасывания окончаний из известного набора (возможно, псевдоосновы за счёт отбрасывания псевдоокончаний). Отличие стемминга от лемматизации можно продемонстрировать на примере слова выходцы: процесс лемматизации определит лемму выходец, в то время как стемминг вернёт псевдооснову выход.

По функциональным возможностям морфологические процессоры делятся на несколько видов:

- выполняющие только лемматизацию или стемминг,
- определяющие часть речи,
- осуществляющие полный морфологический анализ, т. е. лемматизацию и определение всех морфологических характеристик словоформы.
- осуществляющие морфемный анализ, выделяющие морфы, входящие в состав слова.

Введение в морфоанализатор функции морфемного разбора расширяет его применимость. Например, с помощью данной функции может быть реализован поиск семантически близких слов разных частей речи (однокоренных), что может быть полезно в ряде задач компьютерной лингвистики, включая задачу распознавания конструкций по шаблонам. Первые морфологические анализаторы русского языка были простыми и практически не использовали словарной информации. С ростом вычислительных мощностей и объёмов оперативной памяти, а также появлением новых алгоритмов и структур данных стало возможно эффективное использование больших словарей, что значительно улучшило качество реализуемой модели.

Понимание вычислительной системой смысла текста естественного языка представляет собой задачу разбора. Разбор текста выполняется несколькими последовательными операциями: морфологическим разбором, синтаксическим разбором и семантическим разбором. Выполнение задачи синтаксического разбора текста естественного языка представляет собой определение всех синтаксических признаков этих слов, необходимых для семантического разбора. Для решения задач морфологического и синтаксического анализа текста, а так же задач анализа словоизменения применим семантическую нейронную сеть, близкую по свойствам формальной нейронной сети Маккаллока-Питтса.

В подсети извлечения смысла из текста, отдельный нейрон обозначает элементарное понятие, соответствующее этапу обработки, к которому относится данный подслой нейронной сети. Элементарными понятиями являются любые понятия естественного языка с законченным смыслом, такие

как символ, слог, слово, словосочетание, предложение, абзац, весь текст . Об этом говорит сайт https://intellect.icu . Различным этапам обработки соответствуют различные уровни агрегации элементарных понятий, например: символ, слог, слово, словосочетание.

В качестве структуры семантической нейронной сети, выполняющей морфологический и синтаксический разбор, применим синхронизированное линейное дерево. Линейное дерево состоит из подслоев. Каждому синхронизированному подслою соответствует фронт волны обработки. Нейроны первого подслоя соответствуют первой букве слова, второго - второй и так далее. Общее количество подслоев равно максимальному количеству букв в одном слове. Первый подслой состоит из нейронов, распознающих первую букву, второй слой состоит из нейронов распознающих первые две буквы, третий - первые три буквы.

Каждый нейрон имеет одну входную связь с нейроном из предыдущего подслоя, соответствующим предыдущей букве слова, и одну входную связь с нейроном из слоя рецепторов, соответствующим текущей букве. Каждый нейрон может иметь выходную связь с неограниченным количеством нейронов из следующего подслоя обработки. Функции классификации реализуются с помощью агрегирующих подслоев, состоящих из не синхронизированных нейронов. Агрегирующие подслои не синхронизированных нейронов, выполняющих функции дизъюнкции, размещаются между подслоями синхронизированных нейронов, выполняющих функции конъюнкции. В результате получается многослойная структура, в которой после каждого подслоя фронта волны находится подслой агрегирования.

Число нейронов в сети ограниченно, и они имеют конечное число состояний и связей, поэтому слой извлечения смысла в виде синхронизированного линейного дерева можно рассматривать как конечный автомат. Переход из одного состояния в другое происходит при подаче на слой извлечения смысла очередного символа входной последовательности. Пусть одна словарная статья - это группа нейронов, или один нейронный субавтомат в слое извлечения смысла. В случае наличия многозначности, в синхронизированном линейном дереве возбуждаются все словарные статьи и словоформы, соответствующие всем отдельным значением словоформы. Пусть общее число субсостояний словарной статьи равно числу словоформ этой статьи. Пусть каждое субсостояние такого субавтомата представляет собой один возбужденный нейрон.

При этом, в случае одновременного возбуждения двух разных нейронов одного субавтомата будем говорить, что субавтомат имеет одновременно два разных субсостояния. Каждая словарная статья имеет главный нейрон, соответствующий этой статье. Главный нейрон словарной статьи возбужден всегда, когда распознано слово, принадлежащее его словарной статье. Каждой словоформе соответствует отдельный нейрон. Он возбуждается в случае, если словоформа распознана.

В слое извлечения смысла существуют нейроны, не принадлежащие отдельным словарным статьям. Эти нейроны соответствуют признакам словоформ общим для многих словарных статей, таким как род, падеж, число, время

Множество возбужденных нейронов субавтомата соответствует множеству признаков, принадлежащих отдельной словоформе, распознанной субавтоматом. Задача классификации или определения словарной статьи и словоформы по заданной символьной последовательности сводится к прохождению волны возбуждения через слой извлечения смысла и возбуждении соответствующего субавтомата для соответствующей словарной статьи. Задача словоизменения сводится к изменению состояния такого субавтомата из начального состояния - соответствующего словоформе из которой начинается словоизменение в конечное состояние - соответствующее словоформе в которую требуется преобразовать исходную словоформу.

Процесс обработки текста на естественном языке можно разбить на несколько уровней: разбор, анализ и синтез. Уровень разбора определим как функцию преобразования текста на естественном языке из неформализованного вида в формализованное внутреннее представление. Уровень анализа - функция преобразования данных, существующих во внутреннем представлении и вывода на их основе новых данных, так же в формализованном виде. Уровень синтеза - функция формирования ответа на естественном языке, адекватного внутреннему формализованному представлению.

Уровень разбора можно разделить на несколько последовательных операций: морфологический, синтаксический и семантический разбор. Морфологический разбор проводится путем выделения из текста отдельных слов и разбором выделенных слов на морфемы. Операция синтаксического разбора текста естественного языка представляет собой определение всех синтаксических признаков и синтаксических связей этих слов, необходимых для семантического разбора. Семантический разбор требует наличия в памяти системы имитационной модели внешнего мира. Он заключается в окончательном формировании внутреннего формализованного представления текста путем сопоставления фактов, находящихся непосредственно в тексте с знаниями из имитационной модели.

Рассмотрим операции морфологического и синтаксического разбора текста естественного языка. Выделение отдельных слов из текста в электронном виде не представляет трудности, так как в этом случае слова отделяются специальным символом "пробел". Для Русского языка задача морфологического и синтаксического разбора может быть решена с помощью данных содержащихся в грамматическом словаре Русского языка. В нем приводятся информация о словообразовании, словоизменении и определении синтаксических признаков отдельных слов.

Реализация уровня разбора текста требует наличия некоторой формальной модели этого разбора. Согласно теореме Геделя реализация формализма может оказаться проще описания этого формализма. Упоминая

теорему Геделя, фон Нейман пишет: "… В таком случае быстрее сделать что-то, чем описать, быстрее привести схему, чем дать общее описание всех ее функций и всех мыслимых обстоятельств. Очень важно понять, что сеть из формальных нейронов может сделать все, что можно описать словами, и это необычайно упрощает дело при низких уровнях сложности. Но при высоких уровнях сложности это не обязательно будет упрощением. Вполне возможно, что при высоких уровнях сложности ценность представляет собой обратное утверждение этой теоремы Геделя, т. е. она упрощает дело потому, что гарантирует обратное: можно выразить логику на языке этих построений [формальных нейронов], а прямое утверждение может быть и неверным".

Из этого утверждения Неймана следует, что, возможно, в качестве формального представления следует брать не формальные математические модели, а реализацию этих моделей. Одним из вариантов такого формального представления может выступать формальная нейронная сеть. Поэтому можно попытаться воспользоваться формальной нейронной сетью для решения задач морфологического и семантического разбора текста, а так же задачи словоизменения.

В работе описано функционирование формальной нейронной сети Маккаллока-Питтса. Отдельные нейроны в нейронной сети Маккаллока-Питтса представляют собой логические операции и, или и не. В зависимости от результата выполнения логической операции, нейрон имеет состояния возбуждения или покоя, соответствующие логическим значениям истина и ложь.

3.2 Использование библиотек python

Polyglot предлагает обученные модели морфем для создания морфем из слов. Целью проекта Morpho является разработка неконтролируемых методов, основанных на данных, которые выявляют закономерности словообразования в естественных языках. В частности, проект Morpho фокусируется на открытии морфем, которые являются примитивными единицами синтаксиса, мельчайшими индивидуально значимыми элементами в высказываниях языка. Морфемы играют важную роль в автоматической генерации и распознавании языка, особенно в языках, в которых слова могут иметь множество различных флективных форм. Polyglot поддерживает более 100 языков.

Polyglot обучен для определенного списка языков, по 50 тысяч морфем для каждого языка. Просмотреть список языков можно с помощью конструкции:

```
from polyglot.downloader import downloader
print(downloader.supported_languages_table("morph2"))
```

Пример анализа текста

```
from polyglot.text import Text, Word
```

```
words = ["preprocessing", "processor", "invaluable", "thankful", "crossed"]
for w in words:
  w = Word(w, language="en")
  print("{:<20}{}".format(w, w.morphemes))
preprocessing       ['pre', 'process', 'ing']
processor           ['process', 'or']
invaluable          ['in', 'valuable']
thankful            ['thank', 'ful']
crossed             ['cross', 'ed']
```

Если текст не маркирован должным образом, морфологический анализ может предложить разумный способ разделения текста на исходные единицы. Вот, например:

```
blob = "Wewillmeettoday."
text = Text(blob)
text.language = "en"
text.morphemes
WordList([u'We', u'will', u'meet', u'to', u'day', u'.'])
```

4 Задания для выполнения работы

написать парсер для лемматизации текста.

– лемматизировать текст и приписать леммам частеречные теги

– ответить на контрольные вопросы

Для решения задачи можно использовать данные, которые упоминались в лекциях: например, словарь oDict, разметку OpenCorpora и др.

Ввод: предложения вида "токен1 токен2 ... токенN" с расставленными знаками препинания, разделенные переносом строки. Из знаков препинания в предложениях могут содержаться только запятая, точка, вопросительный и восклицательный знаки. Для каждого предложения из входных данных вывод в виде "токен1{лемма1=тег1} токен2{лемма2= тег2} ... токенN{леммаN=тегN}" без исходных знаков препинания. Разделитель между токенами - пробельный символ.

При лемматизации буквы е и ё, а также написание с прописной/строчной буквы признаются равноправными. Частеречные теги должны быть приведены к следующему инвентарю:

существительные (S),

прилагательные (A),

глаголы, в том числе причастия и деепричастия (V),

предлоги (PR),

союзы (CONJ),

сборная категория (ADV), включающая наречия, вводные слова, частицы, междометия.

В оценке не участвуют и могут быть размечены любым образом (в приведенном примере – тег NI) местоимения (включая наречные и предикативные), числительные, а также составные предлоги и союзы (потому что, в течение и т.п.).

Методика выполнения

1. Напишем парсер для лемматизации текста. Для этого создадим новый проект в Pycharm. Необходимо нажать на File → New project, в появившемся окне выбрать путь проекта и интерпретатор (рисунок 3).

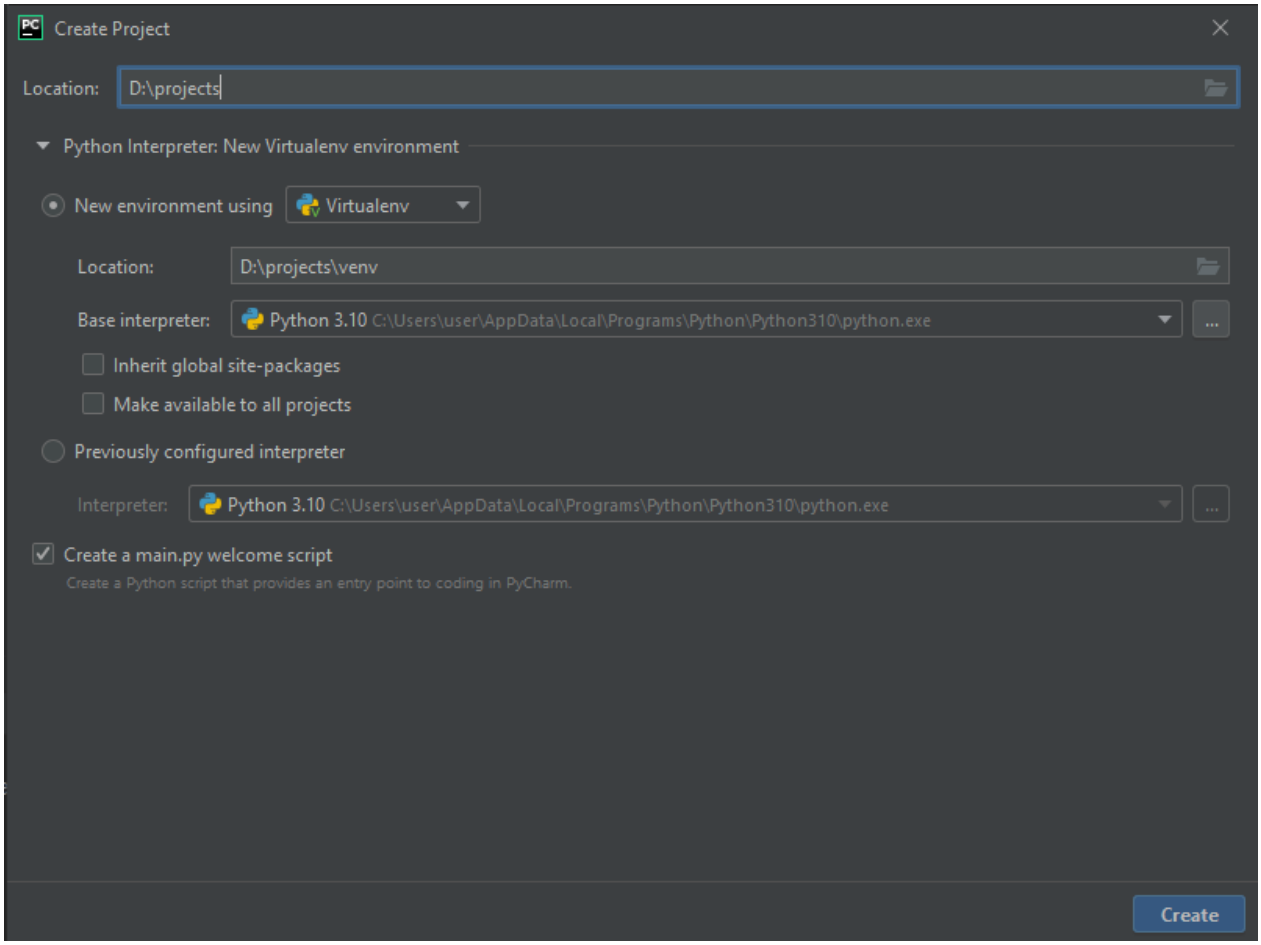

Рисунок 3 – Создание нового проекта в Pycharm

2. В проекте необходимо создать текстовый файл с названием dataset.txt и поместить в него текст для анализа.
3. Импортируем библиотеку и создаем объект класса MorphAnalyzer()

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
```

4. Далее необходимо проанализировать каждое слово текста из файла dataset.txt в переменную. Для этого считываем файл, циклом for итерируем все

строки текста, с помощью split создаем из строки массив слов, и еще одним циклом, но уже вложенным, итерируем все слова в этом массиве.

```
with open("dataset.txt", "r") as a_file:
       for line in a_file:
           stripped_line = line.split()
           for word in stripped_line:
```

5. Далее используем объект morph и анализируем слова

```
word = morph.parse(word)[0].word
normal = morph.parse(word)[0].normal_form
tag = morph.parse(word)[0].tag.POS
```

6. Далее записываем результат каждой итерацией в переменную str, а в конце всех итераций записываем результат в файл

```
str += '{0}{{1}={2}} '.replace('{0}',
word).replace('{1}', normal).replace('{2}', tag if tag
else '')
with open("parsed-text.txt", "w") as text_file:
   text_file.write(str)
```

Листинг 1. Код парсера

```
import pymorphy2
morph = pymorphy2.MorphAnalyzer()
if __name__ == '__main__':
    str = ''
    with open("dataset.txt", "r") as a_file:
        for line in a_file:
            stripped_line = line.split()
            for word in stripped_line:
                word = morph.parse(word)[0].word
                normal = morph.parse(word)[0].normal_form
                tag = morph.parse(word)[0].tag.POS
                str += '{0}{{1}={2}} '.replace('{0}',
word).replace('{1}', normal).replace('{2}', tag if tag
else '')
with open("parsed-text.txt", "w") as text_file:
    text_file.write(str)
```

7. Пример выполнения работы представлен в листинге 1, результат работы программы:

Наиболее{наиболее=ADV} обычен{обычный=A} мелколистный{мелколистный=A} клён{клён=S} полевой{полевой=A} который{который=A} не{не=ADV} смотря{смотреть=V} на{на=PR} такое{такой=A} название{название=S} встречается{встречаться=V} и{и=CONJ} в{в=PR} глубине{глубина=S} тенистых{тенистый=A} ущелий{ущелие=S} Они{они=NPRO} и{и=CONJ} документацию{документация=S} сейчас{сейчас=ADV} оформляют{оформлять=V} только{только=ADV} по{по=PR} страховым{страховой=A} платежам{платёж=S} у{у=PR} них{они=NPRO} отпало{отпасть=V} ФСС{фсс=None} медстраховка{медстраховка=S} и{и=CONJ} отчётность{отчётность=S} у{у=PR} них{они=NPRO} упростилась{упроститься=V} Содержание{содержание=S} лишь{лишь=ADV} мерцает{мерцать=V} сквозь{сквозь=PR} выражение{выражение=S} а{а=CONJ} выражение{выражение=S} лишь{лишь=ADV} намекает{намекать=V} на{на=PR} содержание{содержание=S} Новый{новый=A} босс{босс=S} вызвал{вызвать=V} меня{я=NPRO} на{на=PR} другой{другой=A} день{день=S} в{в=PR} свой{свой=A} кабинет{кабинет=S} Не{не=ADV} стоит{стоить=V} думать{думать=V} что{что=CONJ} достаточно{достаточно=ADV} пойти{пойти=V} к{к=PR} мануальному{мануальный=A} терапевту{терапевт=S} или{или=CONJ} на{на=PR} худой{худой=A} конец{конец=S} к{к=PR} массажисту{массажист=S} чтобы{чтобы=CONJ} устранить{устранить=V} всё{всё=ADV} проблемы{проблема=S} со{с=PR} здоровьем{здоровье=S} раз{раз=S} и{и=CONJ} навсегда{навсегда=ADV} Но{но=CONJ} как{как=CONJ} можно{можно=PRED} прямо{прямо=ADV} доказать{доказать=V} что{что=CONJ} сенсорные{сенсорный=A} способности{способность=S} которые{который=A} отвечают{отвечать=V} миру{мир=S} явлений{явление=S} созданных{создать=V} обществом{общество=S} являются{являться=V} у{у=PR} человека{человек=S} не{не=ADV} врождёнными{врождённый=A} а{а=CONJ} формируются{формироваться=V} прижизненно{прижизненный=A} в{в=PR} результате{результат=S} овладения{овладение=S} этими{этот=A} явлениями?{явлениями?=None} Мы{мы=NPRO} разрабатываем{разрабатывать=V} идеи{идея=S} для{для=PR} сценария{сценарий=S} и{и=CONJ} я{я=NPRO} надеюсь{надеяться=V} что{что=CONJ} получится{получиться=V} забавно{забавно=ADV} Эволюция{эволюция=S} исследуемого{исследовать=V} признака{признак=S} рассматривается{рассматриваться=V} как{как=CONJ} случайное{случайный=A} блуждание{блуждание=S} по{по=PR}

графу{графа=S} из{из=PR} одного{один=A} состояния{состояние=S} в{в=PR} другое{другой=A} согласно{согласно=PR} направлениям{направление=S} дуг{дуга=S} Лишь{лишь=ADV} через{через=PR} несколько{несколько=ADV} дней{день=S} увидел{увидеть=V} я{я=NPRO} его{он=NPRO} среди{среди=PR} гомонящей{гомонить=V} артельки{артелька=S} окружившей{окружить=V} огромадного{огромадный=A} роста{рост=S} и{и=CONJ} веса{вес=S} дядю{дядя=S} Вот{вот=ADV} ты{ты=NPRO} открываешь{открывать=V} справочник{справочник=S} всевозможных{всевозможный=A} образовательных{образовательный=A} учреждений{учреждение=S} города{город=S} Москвы{москва=S} и{и=CONJ} там{там=ADV} безумное{безумный=A} количество{количество=S} страниц{страница=S} сумасшедшее{сумасшедший=A} количество{количество=S} названий{название=S} и{и=CONJ} разобраться{разобраться=V} в{в=PR} этом{это=NPRO} очень{очень=ADV} сложно{сложно=ADV} даже{даже=ADV} если{если=CONJ} классифицировать{классифицировать=V} их{они=NPRO} по{по=PR} географическому{географический=A} признаку{признак=S} всё{всё=ADV} равно{равно=CONJ} в{в=PR} нормальном{нормальный=A} отдалении{отдаление=S} от{от=PR} вашего{ваш=A} дома{дом=S} бывает{бывать=V} иногда{иногда=ADV} до{до=PR} нескольких{несколько=NUMR} десятков{десяток=S} разных{разный=A} школ{школа=S} Но{но=CONJ} тряхнуть{тряхнуть=V} осторожно{осторожно=ADV} без{без=PR} перебора{перебор=S} чтобы{чтобы=CONJ} событие{событие=S} не{не=ADV} выплеснулось{выплеснуться=V} за{за=PR} края{край=S} Благодаря{благодаря=PR} такому{такой=A} разумному{разумный=A} подходу{подход=S} его{он=NPRO} группе{группа=S} удалось{удаться=V} завершить{завершить=V} свою{свой=A} работу{работа=S} раньше{ранний=COMP} других{другой=A}Мимо{мимо=PR} пропускали{пропускать=V} но{но=CONJ} толпа{толпа=S} двигалась{двигаться=V} очень{очень=ADV} медленно{медленно=ADV} подолгу{подолгу=ADV} застывая{застывать=V} на{на=PR} месте{место=S} В{в=PR} благоприятные{благоприятный=A} годы{год=S} сосна{сосна=S} пицундская{пицундский=A} подрастает{подрастать=V} в{в=PR} высоту{высота=S} более{более=ADV} чем{чем=CONJ} на{на=PR} метр{метр=S} Между{между=PR} тем{тем=CONJ} злые{злой=A} языки{язык=S} распространяли{распространять=V} по{по=PR} городу{город=S} слухи{слух=S} оказавшиеся{оказаться=V} правдой{правда=S} что{что=CONJ} тот{тот=A} же{же=ADV} мэр{мэр=S} в{в=PR} прошлом{прошлый=A} году{год=S} самолично{самолично=ADV} ликвидировал{ликвидировать=V} специальную{специальный=A} службу{служба=S} которая{который=A}

занималась{заниматься=V} санитарной{санитарный=A} очисткой{очистка=S} города{город=S}

5 Варианты заданий

1. Если у разработчика в его проекте есть модуль, который подключается к FTP-серверу для загрузки некоторых файлов, то разработчик напишет метод подключения к FTP через URL-адрес FTP и использует учетные данные, такие как имя пользователя и пароль для успешного подключения. Если разработчик использует эти учетные данные в коде и жестко закодирует их в своем файле кода и развернет приложение, оно может работать нормально, работа выполнена! Теперь представьте, что через два месяца пароль для этого FTP-сайта изменен, и разработчик должен снова обновить этот пароль в вашем приложении, чтобы убедиться, что он не нарушает существующую функциональность FTP-соединения. Теперь в этом сценарии разработчик снова изменит код и повторно развернет его. Здесь разработчик должен для небольшого изменения конфигурации использовать последнюю версию кода, внести необходимые изменения, убедиться, что больше ничего не сломается, повторно развернуть приложение и затем протестировать его. И это может быть повторяющейся задачей через 2-3 месяца, когда какая-то конфигурация снова изменится. Что делать, если в приложении есть файл конфигурации, в котором есть пары ключ-значение “Пользователь”: “<имя пользователя>”, “Пароль”: “пароль”, и всякий раз, когда требуется какое-либо изменение, затрагивается только этот файл, и конфигурации обновляются, а не копаются в фактическом коде.
2. Исходя из опыта работы с .NET, мне было немного сложно понять, как разработчик может иметь файл конфигурации в приложении Python, который можно использовать для чтения значений настроек, и при этом не нужно даже прикасаться к коду для обновления или сохранения настроек. Конфигурационные файлы используются для хранения пар ключ-значение или некоторой настраиваемой информации, которая может быть прочитана или доступна в коде и в определенный момент времени. Если эта конфигурация изменится, разработчики могут просто изменить конфигурацию в этом файле конфигурации и не беспокоиться об изменении кода, так как это может потребовать повторной компиляции кода и его развертывания. Не только это, но и использование конфигурационных файлов делает ваши настройки и код более удобными для повторного использования, а также позволяет хранить информацию о настройках централизованно и отдельно. Конечно, конфиденциальная информация, такая как пароли, секреты и сертификаты, должна храниться в большей безопасности и может находиться в облачных хранилищах. Но основные настройки, используемые в приложении, могут быть частью файла конфигурации.
3. Графики знаний существуют уже почти полвека – этот термин был впервые введен в обиход в 1972 году! Долгое время они просто томились в академическом мире, пока Google не анонсировала свой график знаний в 2012

году. С тех пор графики знаний эволюционировали довольно резко, и теперь пути назад нет.За последние 10 лет наблюдается стремительный рост машинного обучения и искусственного интеллекта . Благодаря своей способности внедрять интеллектуальные данные в данные и добавлять контекст, графики знаний используются для того, чтобы сделать МО и ИИ более надежными, заслуживающими доверия и объяснимыми. Вполне естественно, что эти две технологии сближаются, стимулируя рост и необходимость графиков знаний. Графики знаний преобразуют интеллектуальные данные в данные с помощью контекста и взаимосвязей. Графики знаний организуют и предоставляют единое место для поиска соответствующих данных и – что не менее важно – для интерпретации и принятия мер в соответствии с тем, что важно. Связанные данные, обогащенные смыслом, позволяют лучше отвечать на сложные запросы и находить наиболее эффективный путь к бесценной информации.

4. Инкапсуляция — это концепция объектно-ориентированного программирования, которая охватывает детали реализации класса. Почему бы не раскрыть детали реализации? Одна из хороших практик программирования заключается в том, что клиентский код должен иметь доступ только к открытым интерфейсам класса. Пока интерфейс остается прежним, клиентский код не нуждается в изменении в случае изменения реализации. Инкапсуляция также снижает сложность и упрощает процессы отладки. Кроме того, инкапсулированный класс помогает защитить данные и предотвратить неправильное использование, поскольку клиентский код не может получить доступ к защищенной части класса. В дополнение к деталям реализации, элементы данных, как правило, также являются участками кода, которые должны быть защищены. Однако может оказаться целесообразным предоставить общедоступный интерфейс, позволяющий клиентскому коду получать доступ к элементам данных в контексте класса. Этот тип интерфейса обычно называется функцией доступа. Функции доступа обычно бывают двух видов: геттер и сеттер. Получатель — это метод, который вызывается, когда мы обращаемся к элементу данных для чтения. В отличие от геттера, сеттер — это метод, вызываемый для изменения элемента данных.

5. "Тестирование кода" обычно организуется в виде пути, который начинается с модульного тестирования, продолжается интеграцией/сквозным тестированием и заканчивается приемочным тестированием пользователей. В начале этого пути модульное тестирование выполняет две вещи — оно подтверждает, что "тестируемый код" работет правильно при подаче своих входных данных. После этого вы переходите к интеграционному тестированию, где подтверждаете, что "передача" работает правильно (предположительно, только после модульного тестирования "следующая вещь в цепочке обработки", чтобы подтвердить, что она выполняет свою работу правильно). Поскольку API-интерфейсы являются интерфейсом к цепочке обработки, легко спутать модульное тестирование API с интеграционным тестированием всей этой цепочки обработки. Эффективная стратегия тестирования API

предотвращает это. Чтобы продемонстрировать, как эффективно управлять тестированием API, я собираюсь рассмотреть стратегию модульного тестирования API (с использованием [Telerik Test Studio для API), чтобы продемонстрировать, как выглядит эффективная стратегия тестирования API в реальных условиях.

6. Apache Spark - популярная система с открытым исходным кодом для обработки больших объемов данных. Большие данные могут быть большими — возможно, петабайтами. Он также может быть сложным, объединяющим несколько схем и источников. Это также может быть высокоскоростной поток данных. Все эти типы данных выигрывают от обширной параллельной обработки различных частей данных в памяти, чтобы свести к минимуму циклические перемещения в постоянное хранилище. Apache Spark облегчает это, автоматически масштабируя обработку и память для эффективного анализа больших наборов данных. Приложения для Apache Spark включают пакетную обработку фильтрации данных, агрегирование данных и преобразование данных в пригодные для использования наборы данных. Он также может использовать модели машинного обучения и анализа для обработки огромных объемов данных для создания моделей, поиска тенденций и прогнозирования будущих сценариев. Это также полезно для обработки данных в режиме реального времени, которая быстро обрабатывает и анализирует поток данных. Многие другие области применения Apache Spark ограничены только вашим воображением.

6 **Контрольные вопросы:**

1. Три слова в списке имеют одинаковый тип словоизменения, одно - отличается. Выберете это слово.
   a. Табло
   b. Сверло
   c. Чело
   d. Кайло
2. Какие термины относятся к морфологической обработке?
   a. Прокрастинация
   b. Парсинг
   c. Лемминг
   d. Лемматизация
   e. Стемминг
3. Выберете правильный набор граммем для выделенного слова в предложении – «Лично я **ловлю** покемонов не одобряю»
   a. V, несов, пе=непрош, ед, изъяв, 1-л
   b. S, жен, неод=вин, ед
   c. V, несов, пе=ед, пов, 2-л
   d. S, муж, неод=им, ед

4. Замените слова в предложении на пару лемма/часть_речи – «Под круглым прозрачным колпаком из неравномерно утолщающегося книзу стекла сияли торсионные весы»

Практическая работа №2 «Автоматическое построение рефератов текстовых документов»

1 **Цель работы**: изучить методы автоматического построения рефератов текстовых документов.

2 **Порядок выполнения работы**:

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.

3 **Краткие теоретические сведения**

Обобщение - это задача сокращения фрагмента текста до более короткой версии, уменьшения размера исходного текста при одновременном сохранении ключевых информационных элементов и смысла содержания. Поскольку ручное обобщение текста является трудоемкой и, как правило, трудоемкой задачей, автоматизация этой задачи набирает все большую популярность и, следовательно, является сильной мотивацией для академических исследований.

Существуют важные приложения для обобщения текста в различных задачах, связанных с NLP, таких как классификация текста, ответы на вопросы, обобщение юридических текстов, обобщение новостей и генерация заголовков. Кроме того, генерация резюме может быть интегрирована в эти системы в качестве промежуточного этапа, что помогает сократить объем документа.

В эпоху больших данных произошел резкий рост объема текстовых данных из различных источников. Этот объем текста является бесценным источником информации и знаний, которые необходимо эффективно обобщить, чтобы быть полезными. Эта растущая доступность документов потребовала исчерпывающих исследований в области NLP для автоматического обобщения текста. Автоматическое обобщение текста - это задача создания краткого и беглого резюме без какой-либо помощи человека при сохранении смысла исходного текстового документа.

Это очень сложно, потому что, когда мы, люди, обобщаем фрагмент текста, мы обычно читаем его полностью, чтобы развить наше понимание, а затем пишем краткое изложение, выделяя его основные моменты. Поскольку компьютерам не хватает человеческих знаний и языковых возможностей, это делает автоматическое обобщение текста очень сложной и нетривиальной задачей.

Для этой задачи были предложены различные модели, основанные на машинном обучении. Большинство из этих подходов моделируют эту проблему как проблему классификации, которая определяет, следует ли включать предложение в резюме или нет. В других подходах использовалась тематическая информация, Скрытый семантический анализ (LSA), модели

последовательности в последовательности, обучение с подкреплением и состязательные процессы.

В общем, существует два разных подхода к автоматическому обобщению: извлечение и абстракция.

Экстрактивное обобщение извлекает предложения непосредственно из документа на основе функции подсчета очков для формирования последовательного резюме. Этот метод работает путем выделения важных фрагментов текста и объединения частей содержимого для создания сокращенной версии.

Экстрактивное обобщение работает путем выделения важных разделов текста, вырезания и объединения частей содержимого для создания сокращенной версии. Таким образом, они зависят только от извлечения предложений из исходного текста.

Таким образом, они зависят только от извлечения предложений из исходного текста. Большая часть исследований по обобщению сегодня сосредоточена на экстрактивном обобщении, поскольку оно проще и дает естественные грамматические резюме, требующие относительно небольшого лингвистического анализа. Кроме того, извлеченные резюме содержат наиболее важные предложения входных данных, которые могут быть одним документом или несколькими документами.

Типичный поток систем экстрактивного суммирования состоит из:

1. Создает промежуточное представление входного текста, предназначенное для поиска основного содержания. Как правило, он работает путем вычисления показателей TF для каждого предложения в данной матрице.

2. Оценивает предложения на основе представления, присваивая каждому предложению значение, обозначающее вероятность того, что оно будет включено в резюме.

3. Составляет резюме на основе k наиболее важных предложений. В некоторых исследованиях использовался латентный семантический анализ (LSA) для выявления семантически важных предложений.

Методы абстрактного обобщения направлены на создание резюме путем интерпретации текста с использованием передовых методов естественного языка для создания нового более короткого текста, части которого могут не отображаться как часть исходного документа, который передает наиболее важную информацию из исходного текста, требующую перефразирования предложений и включения информации из полного текста для создания резюме, такого как обычно делает реферат, написанный человеком. Фактически, приемлемое абстрактное резюме охватывает основную информацию во входных данных и свободно владеет языком.

Таким образом, они не ограничиваются простым выделением и перестановкой отрывков из оригинального текста.

Абстрактные методы используют преимущества последних разработок в области глубокого обучения. Поскольку это можно рассматривать как задачу сопоставления последовательностей, когда исходный текст должен быть

сопоставлен с целевым резюме, абстрактные методы используют преимущества недавнего успеха моделей последовательности в последовательности. Эти модели состоят из кодера и декодера, где нейронная сеть считывает текст, кодирует его, а затем генерирует целевой текст.

В целом, создание абстрактных резюме является сложной задачей, которая относительно сложнее, чем подходы, основанные на данных, такие как извлечение предложений, и включает в себя сложное языковое моделирование. Таким образом, они все еще далеки от достижения качества составления сводок на уровне человека, несмотря на недавний прогресс в использовании нейронных сетей, вдохновленный прогрессом в области нейронного машинного перевода и моделей последовательности в последовательности.

Недавние исследования показали, что основанные на внимании модели последовательности последовательностей для абстрактного обобщения могут страдать от повторений и семантической неуместности, вызывая грамматические ошибки и недостаточное отражение основной идеи исходного текста. Джуньянг Лин и др. предлагают реализовать блок управления поверх выходов кодера на каждом временном шаге, который представляет собой CNN, который объединяет все выходы кодера, чтобы решить эту проблему.

Основываясь на свертке и самоконтроле Васвани и др., сверточный блок управления устанавливает вентиль для фильтрации исходных аннотаций из кодера RNN, чтобы выбрать информацию, соответствующую глобальному семантическому значению. Другими словами, он уточняет представление исходного контекста с помощью CNN, чтобы улучшить связь представления слова с глобальным контекстом. Их модель способна сократить количество повторений по сравнению с моделью последовательности в последовательности, превосходящей самые современные методы. Исходный код документа можно найти здесь.

Другие методы абстрактного обобщения заимствовали концепции из сети указателей Виньялса и др. для устранения нежелательного поведения моделей последовательности в последовательности. Сеть указателей - это основанная на нейронном внимании архитектура последовательности последовательностей, которая изучает условную вероятность выходной последовательности с элементами, которые являются дискретными маркерами, соответствующими позициям во входной последовательности.

Например, Эбигейл См. и др. представлена архитектура, называемая генератором указателей, которая позволяет копировать слова из входной последовательности путем указания определенных позиций, тогда как генератор позволяет генерировать слова из фиксированного словаря из 50 тыс. слов. Архитектуру можно рассматривать как баланс между экстрактивным и абстрактным подходами.

Чтобы преодолеть проблемы с повторением, в статье адаптирована модель охвата Tu и др., которая была предложена для преодоления недостаточного охвата исходных слов в моделях нейронного машинного перевода. В частности, Эбигейл Си и др. определили гибкую потерю покрытия,

чтобы наказывать за повторное посещение одних и тех же мест, наказывая только за перекрытие между каждым распределением внимания и охватом до текущего временного шага, помогая предотвратить повторное внимание.

Другие исследования в области абстрактного обобщения заимствовали концепции из области обучения с подкреплением (RL) для повышения точности модели. Например, Чен и др. предложили гибридную экстрактивно-абстрактную архитектуру, использующую две нейронные сети иерархическим способом, которая выбирает основные предложения с помощью экстрактора, управляемого RL, из источника, а затем переписывает их абстрактно для создания резюме.

Другими словами, модель имитирует, как люди обобщают длинные документы, сначала используя агент извлечения для выбора основных предложений или основных моментов, а затем использует абстрактор — модель- сеть кодировщика-выравнивателя—декодера для перезаписи каждого из этих извлеченных предложений. Для обучения экстрактора доступным парам "документ-резюме" в модели используется основанное на политике обучение с подкреплением (RL) с метрическими вознаграждениями на уровне предложений для подключения сетей экстрактора и абстрактора и изучения значимости предложений.

Агент извлечения представляет собой кодировщик сверточных предложений, который вычисляет представления для каждого предложения на основе входных встроенных векторов слов. Кроме того, кодировщик RNN вычисляет контекстно-зависимое представление, а затем декодер RNN выбирает предложение на временном шаге t. Как только предложение выбрано, контекстно-зависимое представление будет передано в декодер в момент времени t + 1.

Таким образом, метод включает в себя преимущества абстрактного подхода, заключающиеся в кратком переписывании предложений и генерации новых слов из полного словаря, в то время как использует промежуточное экстрактивное поведение для улучшения качества, скорости и стабильности общей модели. Автор утверждал, что обучение модели происходит в 4 раза быстрее, чем в предыдущей версии. Как исходный код, так и лучшие предварительно подготовленные модели были выпущены для продвижения будущих исследований.

В других недавних исследованиях предлагалось использовать комбинацию состязательных процессов и обучения с подкреплением для абстрактного обобщения. Примером может служить Liu и др. (2017), в работе которых предлагается состязательная структура для совместного обучения генеративной модели и дискриминационной модели, аналогичной Goodfellow и др. (2014). В этой структуре генеративная модель использует исходный текст в качестве входных данных и генерирует резюме с использованием обучения с подкреплением, чтобы оптимизировать генератор для получения резюме с высокой наградой. Кроме того, модель дискриминатора пытается отличить сводки основной истины от сводок, сгенерированных генератором.

Дискриминатор реализован в виде классификатора текста, который учится классифицировать сгенерированные резюме как машинные или созданные человеком, в то время как процедура обучения генератора заключается в максимизации вероятности ошибки дискриминатора. Идея заключается в том, что этот состязательный процесс в конечном итоге может позволить генератору генерировать правдоподобные и высококачественные абстрактные резюме.

Gensim - очень удобная библиотека python для выполнения задач NLP. Процесс обобщения текста с использованием библиотеки gensim основан на алгоритме TextRank

TextRank - это метод экстрактивного обобщения. Он основан на концепции, что слова, которые встречаются чаще, являются значимыми. Следовательно, предложения, содержащие очень часто встречающиеся слова, очень важны.

Основываясь на этом, алгоритм присваивает баллы каждому предложению в тексте. Предложения с самым высоким рейтингом попадают в резюме.

Рассмотрим приведенную ниже статью о нездоровой пище, которая должна быть обобщена.

original_text = Нездоровая пища приятна на вкус, поэтому ее в основном любят все люди любой возрастной группы, особенно дети и школьники. Обычно они ежедневно просят нездоровую пищу, потому что так их учили родители с детства. Родители никогда не обсуждали с ними вредное воздействие нездоровой пищи на здоровье. Согласно исследованиям ученых, было установлено, что нездоровая пища оказывает негативное влияние на здоровье во многих отношениях. Как правило, это жареная пища, которую можно найти на рынке в пакетах. В них становится много калорий, много холестерина, мало полезных питательных веществ, много минералов натрия, много сахара, крахмала, вредных жиров, недостатка белка и пищевых волокон. Обработанная и нездоровая пища является средством быстрого и нездорового увеличения веса и негативно влияет на весь организм на протяжении всей жизни. Это позволяет человеку набрать избыточный вес, который называется ожирением. Нездоровая пища приятна на вкус и хорошо выглядит, однако не удовлетворяет здоровым потребностям организма в калориях. Некоторые продукты, такие как картофель фри, жареная пища, пицца, гамбургеры, конфеты, безалкогольные напитки, выпечка, мороженое, печенье и т.д., Являются примером продуктов с высоким содержанием сахара и высоким содержанием жира. По данным Центров по контролю и профилактике заболеваний, установлено, что дети и особенно, употребляющие нездоровую пищу, более склонны к диабету 2 типа. При диабете 2 типа наш организм становится неспособным регулировать уровень сахара в крови. Риск заболеть этим заболеванием возрастает по мере того, как человек становится более тучным или с избыточным весом. Это увеличивает риск развития почечной недостаточности. Ежедневное употребление нездоровой пищи приводит нас к

дефициту питательных веществ в организме, потому что это недостаток необходимых питательных веществ, витаминов, железа, минералов и пищевых волокон. Он увеличивает риск сердечно-сосудистых заболеваний, потому что богат насыщенными жирами, натрием и плохим холестерином. Диета с высоким содержанием натрия и плохого холестерина повышает кровяное давление и перегружает работу сердца. Тот, кто любит нездоровую пищу, больше рискует набрать лишний вес, стать толще и нездоровее. Нездоровая пища содержит большое количество углеводов, которые повышают уровень сахара в крови и делают человека более вялым, сонным, менее активным и бдительным. Рефлексы и чувства людей, употребляющих эту пищу, притупляются день ото дня, поэтому они ведут более сидячий образ жизни. Нездоровая пища является источником запоров и других заболеваний, таких как диабет, сердечные заболевания, закупорка артерий, сердечный приступ, инсульты и т.д. Из-за плохого питания. Нездоровая пища - самый простой способ набрать нездоровый вес. Количество жиров и сахара в пище заставляет вас быстро набирать вес. Однако это не здоровый вес. В нем больше жиров и холестерина, которые окажут вредное воздействие на ваше здоровье. Нездоровая пища также является одной из главных причин роста ожирения в настоящее время.Эта еда только выглядит и имеет приятный вкус, кроме этого, в ней нет никаких положительных моментов. Количество калорий, необходимое вашему организму для поддержания физической формы, не удовлетворяется этой пищей. Например, такие продукты, как картофель фри, гамбургеры, конфеты и печенье, содержат большое количество сахара и жиров. Таким образом, это может привести к длительным заболеваниям, таким как диабет и высокое кровяное давление. Это также может привести к почечной недостаточности. Прежде всего, вы можете получить различные недостатки в питании, если не потребляете необходимые питательные вещества, витамины, минералы и многое другое. Вы становитесь склонны к сердечно-сосудистым заболеваниям из-за потребления плохого холестерина и жира плюс натрия. Другими словами, все это мешает работе вашего сердца. Кроме того, нездоровая пища содержит более высокий уровень углеводов. Это мгновенно повысит уровень сахара в крови. Это приведет к вялости, бездействию и сонливости. Рефлекс человека со временем притупляется, и он ведет малоподвижный образ жизни. Что еще хуже, нездоровая пища также закупоривает ваши артерии и увеличивает риск сердечного приступа. Поэтому этого следует избегать в первую очередь, чтобы спасти свою жизнь от разрушения.Главная проблема нездоровой пищи заключается в том, что сейчас люди не осознают ее пагубных последствий. Когда придет время, будет уже слишком поздно. Самое главное, проблема в том, что это не влияет на вас мгновенно. Это работает на вас сверхурочно; рано или поздно вы столкнетесь с последствиями. Таким образом, лучше остановиться сейчас.Вы можете избежать нездоровой пищи, поощряя своих детей с раннего возраста есть зеленые овощи. Их вкусовые рецепторы должны быть развиты таким образом, чтобы они находили здоровую пищу вкусной. Более того, попробуйте все

перепутать. Не подавайте один и тот же зеленый овощ ежедневно в одном и том же стиле. Включайте различные виды здоровой пищи в свой рацион по разным рецептам. Это поможет им попробовать еду дома, а не увлекаться нездоровой пищей.Короче говоря, не лишайте их этого полностью, так как это не поможет. Дети найдут тот или иной способ получить это. Убедитесь, что вы даете им нездоровую пищу в ограниченных количествах и в здоровые периоды времени.'

После импорта пакета gensim первым шагом является импорт summary из gensim.summarization. Это встроенная функция, которая реализует TextRank.

```
# Importing package and summarizer
import gensim
from gensim.summarization import summarize
```

Затем передайте текстовый корпус в качестве входных данных для функции автореферирования.

```
# Passing the text corpus to summarizer
short_summary = summarize(original_text)
print(short_summary)
```

Полученный текст:

В них становится много калорий, много холестерина, мало полезных питательных веществ, много минералов натрия, много сахара, крахмала, вредных жиров, недостатка белка и пищевых волокон. Обработанная и нездоровая пища является средством быстрого и нездорового увеличения веса и негативно влияет на весь организм на протяжении всей жизни.

Нездоровая пища приятна на вкус и хорошо выглядит, однако не удовлетворяет здоровым потребностям организма в калориях. По данным Центров по контролю и профилактике заболеваний, установлено, что дети и особенно, употребляющие нездоровую пищу, более склонны к диабету 2 типа. Ежедневное употребление нездоровой пищи приводит нас к дефициту питательных веществ в организме, потому что это недостаток необходимых питательных веществ, витаминов, железа, минералов и пищевых волокон.

Он увеличивает риск сердечно-сосудистых заболеваний, потому что богат насыщенными жирами, натрием и плохим холестерином. Диета с высоким содержанием натрия и плохого холестерина повышает кровяное давление и перегружает работу сердца. Тот, кто любит нездоровую пищу, больше рискует набрать лишний вес, стать толще и нездоровее. Нездоровая пища содержит большое количество углеводов, которые повышают уровень сахара в крови и делают человека более вялым, сонным, менее активным и бдительным. Например, такие продукты, как картофель фри, гамбургеры, конфеты и печенье, содержат большое количество сахара и жиров.

Автореферат получился слишком большой, но в библиотеке genism можно контролировать некоторые параметры:

1. ratio – он может принимать значения от 0 до 1. Он представляет собой долю резюме по сравнению с исходным текстом.
2. word_count – определяет количество слов в резюме

```
# Summarization by ratio
summary_by_ratio=summarize(original_text,ratio=0.1)
print(summary_by_ratio)
```

Полученный текст:

В них становится много калорий, много холестерина, мало полезных питательных веществ, много минералов натрия, много сахара, крахмала, вредных жиров, недостатка белка и пищевых волокон.Обработанная и нездоровая пища является средством быстрого и нездорового увеличения веса и негативно влияет на весь организм на протяжении всей жизни.

Ежедневное употребление нездоровой пищи приводит нас к дефициту питательных веществ в организме, потому что это недостаток необходимых питательных веществ, витаминов, железа, минералов и пищевых волокон.

Он увеличивает риск сердечно-сосудистых заболеваний, потому что богат насыщенными жирами, натрием и плохим холестерином.Диета с высоким содержанием натрия и плохого холестерина повышает кровяное давление и перегружает работу сердца.

В приведенном выше выводе вы можете заметить, что только 10% исходного текста взято в качестве резюме.

```
# Summarization by word count
summary_by_word_count=summarize(article_text,word_count=30)
print(summary_by_word_count)
```

Полученный текст:

В них становится много калорий, много холестерина, мало полезных питательных веществ, много минералов натрия, много сахара, крахмала, вредных жиров, недостатка белка и пищевых волокон.

Реферирование и аннотирование текста являются сложными видами интеллектуальной деятельности. Составление человеком рефератов или аннотаций занимает много времени. Это приводит к тому, что до ученых, педагогов, инженеров и других специалистов новейшая информация (особенно зарубежная) доходит очень медленно, что, в свою очередь, ведет к повторению

в разных странах и в пределах одной страны одних и тех же исследований, более позднему применению новейших методик, технологий, процессов. Чтобы как-то избежать этого, для составления рефератов и аннотаций применяют современные компьютеры.

Составление реферата или аннотации текста с помощью компьютера называется автоматическим реферированием или аннотированием.

При выполнении работы по составлению реферата или аннотации человеком (референтом) обычно выделяют три этапа:

1) подготовительный — референт определяет тематическую направленность текста и пытается понять и осмыслить документ в целом;

2) аналитический — референт делит текст на некоторые фрагменты (абзацы, аспекты и т.п.). Каждый фрагмент внимательно изучается, в нем выделяют основные смысловые единицы (предложения, словосочетания, слова). Данный этап заканчивается составлением плана будущих реферата или аннотации;

3) этап непосредственного построения реферата или аннотации — выделенные ранее смысловые единицы (их комбинации или преобразования) располагаются в единый вторичный текст в соответствии с планом реферата или аннотации.

В качестве основных смысловых единиц, выделяемых из исходного текста на 2-м этапе, могут выступать:

1) целые ключевые предложения;

2) ключевые словосочетания и слова.

Ключевое (опорное) слово — это термин, относящийся к основному содержанию текста и повторяющийся в нем несколько раз (с учетом всех возможных синонимов).

Ключевое словосочетание — это сочетание слов, среди которых есть одно или несколько ключевых.

Ключевым предложением считается предложение, содержащее два и более ключевых слова или ключевых словосочетания.

Составление плана будущих реферата или аннотации заключается в выделении некоторых смысловых ориентиров, которые на 3-м этапе будут развернуты более подробно. В качестве таких ориентиров выступают:

1) основные темы и подтемы исходного текста;

2) основные аспекты исследования;

3) основные ключевые предложения, словосочетания и слова.

Создаваемый на 3-м этапе реферат или аннотация содержат выделенные ранее смысловые единицы. В качестве смысловых единиц реферата могут выступать:

1) полные (без изменения) ключевые предложения исходного текста;

2) перефразированные ключевые предложения исходного текста;

3) предложения, составленные из ключевых слов или словосочетаний исходного текста с помощью специальных связующих элементов;

4) предложения, обобщающие несколько предложений исходного текста (не обязательно ключевых).

При перефразировании применяются различные лексико-грамматические явления: использование синонимов, конверсивов, замен по принципу «вид — род», «часть — целое» и т.п.

При получении новых предложений из ключевых слов и словосочетаний исходного текста чаще всего используют различные логико-смысловые скрепы, например, потому что, в то время как, поэтому, вследствие и т.п.

В обобщающих предложениях исходный текст передается совершенно другими словами. В них то же самое содержание излагается в более кратком виде.

Смысловыми единицами аннотации могут быть:

1) ключевые слова или словосочетания исходного текста с предшествующими им специальными фразами — реляторами типа: «В статье рассматриваются следующие вопросы:...», «Книга посвящена следующим проблемам: ...» и т.п.;

2) специальные предложения, содержащие оценочные элементы: «Рассматривается важная проблема...», «Статья посвящена актуальной теме...» и т.д.;

3) специальные предложения, содержащие клише, т.е. специализированные словесные штампы, фиксирующие внимание читателя на определенных аспектах содержания: «Недостаток... заключается», «Цель публикации...», «Ставится задача...», «Делается попытка...» и т.д.

Следующий важный вопрос, который необходимо рассмотреть, связан с тем, как человек выбирает из текста ключевые предложения, словосочетания и слова. Это делается, как уже отмечалось, на 2-м этапе общего процесса составления вторичного документа. Читая текст повторно (первый раз он читается на подготовительном этапе) или в третий раз, человек мысленно выделяет в нем три типа единиц (предложений, словосочетаний, слов):

1) единицы, которые обязательно должны быть включены в реферат или аннотацию. Такие единицы отражают новые идеи, гипотезы, новые методы, явления, процессы, новые результаты, т.е. все новое и оригинальное, что есть в исходном документе. Это, по существу, и есть основные смысловые единицы текста (ключевые предложения, словосочетания и слова);

2) единицы, которые отражают фактические данные: параметры изделий, процессов, методов и т.д. Такие единицы не являются принципиально новыми;

3) единицы, которые аргументируют и иллюстрируют единицы первых двух типов.

Единицы первого уровня обязательно используются при составлении реферата. Из единиц второго уровня используются лишь некоторые (в зависимости от типа реферата или его потребителя). Третья группа единиц изредка переносится в реферат в обобщенном виде.

Если поручить составление реферата или аннотации компьютеру, то, очевидно, его надо научить выполнять те же действия, которые осуществляет человек. Компьютер должен уметь:

1) находить в тексте ключевые слова, словосочетания и предложения;

2) находить в тексте менее значимые единицы;

3) составлять из текстовых единиц двух первых типов смысловые единицы реферата или аннотации;

4) составлять из таких единиц текст реферата или аннотации. Говоря о двух последних «умениях» компьютера, необходимо помнить, что почти во всех существующих системах автоматического реферирования в качестве основных смысловых единиц реферата выступают ключевые предложения или ключевые словосочетания и слова исходного текста. Первые в их последовательной совокупности (в том порядке, в котором они идут в исходном тексте) образуют текст (квазитекст) реферата. Второй тип смысловых единиц (ключевые словосочетания и слова) используется компьютером для построения так называемых табличных рефератов.

При составлении с помощью компьютера аннотации также используются как ключевые предложения (в том виде, что и при составлении реферата), так и ключевые слова и словосочетания. Последние перечисляются вслед за реляторами вида: «В статье рассматриваются следующие вопросы:...», «Книга посвящена следующим проблемам: ...», «Статья раскрывает следующие понятия: ...» и т.д.

По способам выделения из исходных текстов ключевых словосочетаний и предложений (первые два «умения» компьютера) различают несколько методов автоматического реферирования и аннотирования текстов. Наиболее известны следующие три группы методов:

1) статистические;

2) позиционные;

3) логико-семантические.

Суть статистической группы методов заключается в том, что:

1) ключевыми словами считаются такие знаменательные слова текста, которые с учетом всех синонимов встречаются в тексте наибольшее число раз;

2) ключевым предложением считается предложение текста, которое:

а) имеет несколько ключевых слов;

б) содержит ключевые слова на небольшом расстоянии друг от друга.

Принадлежность слова, словосочетания или предложения к числу ключевых определяется специальными статистическими коэффициентами.

В позиционных методах автоматического реферирования и аннотирования ключевым предложением считается предложение, входящее в заголовок, подзаголовок, начало или конец какой-то части текста или всего текста. Такие предложения, как правило, содержат информацию о целях, методах, выводах и результатах исследования, описанного в первичном документе. Важность тех или иных предложений с указанной точки зрения

определяется экспертами путем изучения семантической структуры первичных документов определенного типа.

Логико-семантические методы опираются на исследование структуры и семантики текстов. Существует несколько вариантов этих методов, но цель их одна — выделить из конкретного текста предложения с наибольшим функциональным весом. Величина эта зависит от многих факторов: наличия в исследуемом предложении специальных семантически значимых слов, связи этого предложения с другими предложениями текста, синтаксического типа самого предложения и т.д.

Глубинный способ формирования рефератов предполагает наличие методов синтаксического или семантического разбора предложений. В первом случае используются деревья синтаксического разбора. Процедуры автоматического реферирования манипулируют непосредственно деревьями, выполняя перегруппировку и сокращение ветвей на основании соответствующих критериев. Такое упрощение обеспечивает построение реферата — структурную выжимку исходного текста. Во втором случае на этапе анализа также выполняется синтаксический разбор текста, но синтаксические деревья не порождаются, а формируются семантические структуры, которые накапливаются в виде концептуальных подграфов в базах знаний или тезаурусах. В частности, известны модели, позволяющие производить реферирование текстов на основе психологических ассоциаций сходства и контраста. В базах знаний избыточная и не имеющая прямого отношения к тексту информация устраняется путем отсечения некоторых подграфов. Затем информация подвергается агрегированию методом слияния оставшихся графов или их обобщения. Для осуществления этих преобразований выполняются манипуляции логическими предположениями, выделяются определенные шаблоны в текстовой базе знаний. В результате преобразования формируется концептуальная структура текста в виде аннотации. Многоуровневое структурирование текста с использованием семантических методов позволяет подходить к решению задачи реферирования различными путями.

1. Удаление малозначащих смысловых единиц. Преимуществом метода является гарантированное сохранение значащей информации, недостатком — низкая степень сжатия, т. е. сокращения объема реферата по сравнению с первичными документами.

2. Сокращение смысловых единиц — замена их основной лексической единицей, выражающей основной смысл. 3. Гибридный способ, заключающийся в уточнении реферата с помощью статистических методов, с использованием семантических классов, особенностей контекста и синонимических связей. Некоторые авторы выделяют пять различных подходов к автореферированию (рис. 1):

– статистический подход;

– когерентный подход;

– алгебраический подход;

– графовый подход;
– подход, основанный на машинном обучении.

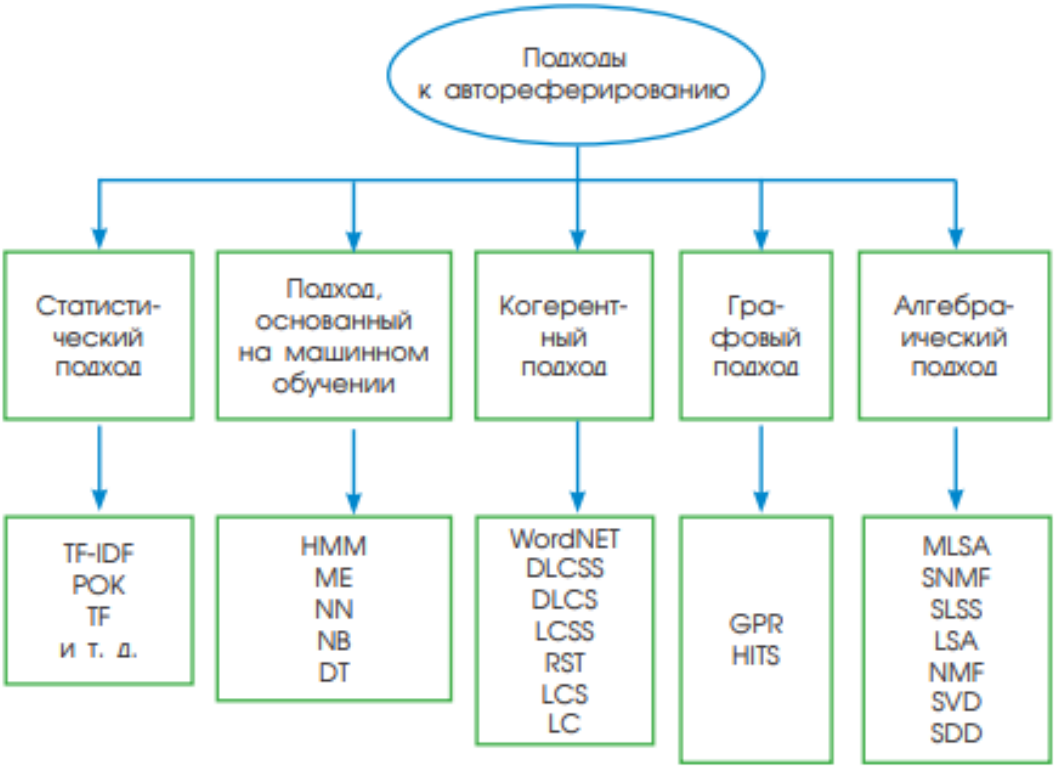

Рисунок 1 – Классификация подходов автореферирования текста

Статистический подход - этот подход очень прост и часто используется для извлечения ключевых слов из документов. Для этого подхода нет предопределенного набора данных. Чтобы извлечь ключевые слова из документов, он использует несколько статистических характеристик документа, таких как частота слова (TF), временная частота обратных документов (TF-IDF), позиция ключевого слова (POK) и т. д.

Когерентный подход - этот подход в основном касается отношений согласованности между словами. Сопряженными отношениями между элементами в тексте характеризуются ссылка, эллипсис, замещение, союз и лексическая когерентность, а также лексическая цепочка слова (LC), WordNet (WN), оценка лексической цепочки (LCS), оценка прямой лексической цепочки (DLCS), оценка диапазона лексической цепочки (LCSS), оценка диапазона прямой лексической цепочки (DLCSS).

Алгебраический подход - в этом подходе используются алгебраические теории, а именно матрица, транспонирование матрицы, собственные векторы и т. д. Существует много алгоритмов, используемых для обобщения текста на основе алгебраического подхода, например, латентный семантический анализ (LSA), мета-латентный семантический анализ (MLSA), факторизация симметричных неотрицательных матриц (SNMF), семантический анализ уровня

предложений (SLSS), факторизация неотрицательных матриц (NMF), сингулярное разложение (SVD), полудискретное разложение (SDD).

Графовый подход заключается в том, что фрагменты текста (слова, предложения, абзацы, в нашем случае — ЭДЕ) описываются в виде вершин графа, а отношения между вершинами (например, семантические отношения) обозначаются ребрами. Кроме того, для обнаружения в тексте важных фрагментов используются такие популярные методы, основанные на графах, как поиск гиперссылок с индуцированными темами (HITS) и Google PageRank (GPR).

Машинное обучение — подход, характерной чертой которого является не прямое решение задачи, а обучение в процессе применения решений множества сходных задач. Для обучения нужен размеченный набор данных. Выходом алгоритма обучения является функция, аппроксимирующая неизвестную (восстанавливаемую) зависимость. Существует несколько популярных подходов к компьютерному обучению: метод Байеса (NB), деревья решений (DT), скрытая марковская модель (HMM), максимальная энтропия (ME), нейронные сети (NN), метод опорных векторов (SVM). В такой классификации статистический и алгебраический подходы могут считаться экстракцией, когерентный подход и подход, основанный на машинном обучении, — абстракцией. Графовый подход является гибридным. На международном рынке представлено множество программных продуктов, которые позволяют создавать авторефераты. Ориентированы они преимущественно на документы, содержащие текст на английском языке.

Тексты рефератов, как уже говорилось ранее, могут полностью состоять из предложений, извлеченных из исходного текста (полная экстракция), и представлять собой комбинацию фрагментов исходного и нового текста, возможно, даже в рамках одного предложения (сочетание экстракции и абстракции), а также не содержать предложений исходного текста вообще (полная абстракция). В данной работе метод экстракции использовался для формирования квазиреферата.

В рамках квазиреферирования выделяют три основных направления, которые в современных системах применяются совместно:

– статистические методы, основанные на оценке информативности разных элементов текста по частоте появления, которая служит основным критерием информативности слов, предложений или фраз;

– позиционные методы, которые опираются на предположение о том, что информативность элемента текста зависит от его позиции в документе;

– индикаторные методы, основанные на оценке элементов текста, исходя из наличия в них специальных слов и словосочетаний —маркеров важности, которые характеризуют их содержательную значимость. После выявления определенного (задаваемого, как правило, коэффициентом необходимого сжатия) количества текстовых блоков с наивысшими весовыми коэффициентами они объединяются для построения квазиреферата.

4 Задания для выполнения работы

– разработать программу для автоматического построения рефератов текстовых документов

– ответить на контрольные вопросы

Методика выполнения

Максимальный размер каждого из рефератов -- 300 символов (включая пробельные). Если размер реферата превышает указанный порог, то будут оцениваться только первые 300 символов. Тривиальное решение (первые 300 символов документа) допускается, но не приветствуется.

Оценка: ROUGE-2 -- близость набору вручную составленных рефератов на основе биграмма слов (значение от 0 до 1).

Sample Input:

["Первый текст...", "Второй текст..."]

Sample Output:

["Реферат первого текста...", "Реферат второго текста..."]

1. Для выполнения практического задания создать программу на языке Python(рисунок 3), которая создает автоматический реферат для входного документа.
2. Данный язык необходимо выбрать, так как его инструментарий содержит все необходимые инструменты для автоматического реферирования документов.
3. В частности, использовать пакет textrank и инструменты, входящие в пакет для обработки естественного языка на Python – nltk.
4. В результате автоматического реферирования получить результат не хуже, чем на рисунке 2.

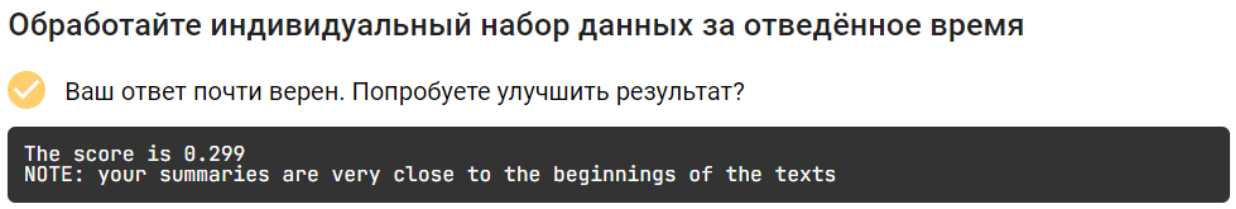

Рисунок 2 – Результат выполнения практического задания

```
from itertools import combinations
import nltk
from nltk.tokenize import sent_tokenize, RegexpTokenizer
from nltk.stem.snowball import RussianStemmer
import networkx as nx

def similarity(s1, s2):
    if not len(s1) or not len(s2):
        return 0.0
    return len(s1.intersection(s2))/(1.0 * (len(s1) + len(s2)))

# Выдает список предложений отсортированных по значимости
def textrank(text):
    sentences = sent_tokenize(text)
    tokenizer = RegexpTokenizer(r'\w+')
    lmtzr = RussianStemmer()
    words = [set(lmtzr.stem(word) for word in tokenizer.tokenize(sentence.lower()))
             for sentence in sentences]
    pairs = combinations(range(len(sentences)), 2)
    scores = [(i, j, similarity(words[i], words[j])) for i, j in pairs]
    scores = filter(lambda x: x[2], scores)
    g = nx.Graph()
    g.add_weighted_edges_from(scores)
    pr = nx.pagerank(g)
    return sorted(((i, pr[i], s) for i, s in enumerate(sentences) if i in pr), key=lambda x: pr[x[0]], reverse=True)

# Сокращает текст до нескольких наиболее важных предложений
def sumextract(text, n=5):
    tr = textrank(text)
    top_n = sorted(tr[:n])
    return ' '.join(x[2] for x in top_n)


#загрузить из json
with open('C:\\Python Data\\input.json', 'r', encoding='utf-8') as fh: #открываем файл на чтение
    data = json.load(fh) #загружаем из файла данные в словарь data

for text in data:
    print(sumextract(text))

#сохранить в json
with open('C:\\Python Data\\data.json', 'w', encoding='utf-8') as fh: #открываем файл на запись
    fh.write(json.dumps(data, ensure_ascii=False)) #преобразовываем словарь data в unicode-строку и записываем в файл
```

Рисунок 3 – Программа для автоматического реферирования

5 Варианты заданий

1. Если у разработчика в его проекте есть модуль, который подключается к FTP-серверу для загрузки некоторых файлов, то разработчик напишет метод подключения к FTP через URL-адрес FTP и использует учетные данные, такие как имя пользователя и пароль для успешного подключения. Если разработчик использует эти учетные данные в коде и жестко закодирует их в своем файле кода и развернет приложение, оно может работать нормально, работа выполнена! Теперь представьте, что через два месяца пароль для этого FTP-сайта изменен, и разработчик должен снова обновить этот пароль в вашем приложении, чтобы убедиться, что он не нарушает существующую функциональность FTP-соединения. Теперь в этом сценарии разработчик снова изменит код и повторно развернет его. Здесь разработчик должен для небольшого изменения конфигурации использовать последнюю версию кода, внести необходимые изменения, убедиться, что больше ничего не сломается, повторно развернуть приложение и затем протестировать его. И это может быть повторяющейся задачей через 2-3 месяца, когда какая-то конфигурация снова изменится. Что делать, если в приложении есть файл конфигурации, в котором есть пары ключ-значение “Пользователь”: “<имя пользователя>”, “Пароль”: “пароль”, и всякий раз, когда требуется какое-либо изменение, затрагивается

только этот файл, и конфигурации обновляются, а не копаются в фактическом коде.

2. Исходя из опыта работы с .NET, мне было немного сложно понять, как разработчик может иметь файл конфигурации в приложении Python, который можно использовать для чтения значений настроек, и при этом не нужно даже прикасаться к коду для обновления или сохранения настроек. Конфигурационные файлы используются для хранения пар ключ-значение или некоторой настраиваемой информации, которая может быть прочитана или доступна в коде и в определенный момент времени. Если эта конфигурация изменится, разработчики могут просто изменить конфигурацию в этом файле конфигурации и не беспокоиться об изменении кода, так как это может потребовать повторной компиляции кода и его развертывания. Не только это, но и использование конфигурационных файлов делает ваши настройки и код более удобными для повторного использования, а также позволяет хранить информацию о настройках централизованно и отдельно. Конечно, конфиденциальная информация, такая как пароли, секреты и сертификаты, должна храниться в большей безопасности и может находиться в облачных хранилищах. Но основные настройки, используемые в приложении, могут быть частью файла конфигурации.

3. Графики знаний существуют уже почти полвека – этот термин был впервые введен в обиход в 1972 году! Долгое время они просто томились в академическом мире, пока Google не анонсировала свой график знаний в 2012 году. С тех пор графики знаний эволюционировали довольно резко, и теперь пути назад нет.За последние 10 лет наблюдается стремительный рост машинного обучения и искусственного интеллекта . Благодаря своей способности внедрять интеллектуальные данные в данные и добавлять контекст, графики знаний используются для того, чтобы сделать МО и ИИ более надежными, заслуживающими доверия и объяснимыми. Вполне естественно, что эти две технологии сближаются, стимулируя рост и необходимость графиков знаний. Графики знаний преобразуют интеллектуальные данные в данные с помощью контекста и взаимосвязей. Графики знаний организуют и предоставляют единое место для поиска соответствующих данных и – что не менее важно – для интерпретации и принятия мер в соответствии с тем, что важно. Связанные данные, обогащенные смыслом, позволяют лучше отвечать на сложные запросы и находить наиболее эффективный путь к бесценной информации.

4. Инкапсуляция — это концепция объектно-ориентированного программирования, которая охватывает детали реализации класса. Почему бы не раскрыть детали реализации? Одна из хороших практик программирования заключается в том, что клиентский код должен иметь доступ только к открытым интерфейсам класса. Пока интерфейс остается прежним, клиентский код не нуждается в изменении в случае изменения реализации. Инкапсуляция также снижает сложность и упрощает процессы отладки. Кроме того, инкапсулированный класс помогает защитить данные и предотвратить

неправильное использование, поскольку клиентский код не может получить доступ к защищенной части класса. В дополнение к деталям реализации, элементы данных, как правило, также являются участками кода, которые должны быть защищены. Однако может оказаться целесообразным предоставить общедоступный интерфейс, позволяющий клиентскому коду получать доступ к элементам данных в контексте класса. Этот тип интерфейса обычно называется функцией доступа. Функции доступа обычно бывают двух видов: геттер и сеттер. Получатель — это метод, который вызывается, когда мы обращаемся к элементу данных для чтения. В отличие от геттера, сеттер — это метод, вызываемый для изменения элемента данных.

5. "Тестирование кода" обычно организуется в виде пути, который начинается с модульного тестирования, продолжается интеграцией/сквозным тестированием и заканчивается приемочным тестированием пользователей. В начале этого пути модульное тестирование выполняет две вещи — оно подтверждает, что "тестируемый код" работет правильно при подаче своих входных данных. После этого вы переходите к интеграционному тестированию, где подтверждаете, что "передача" работает правильно (предположительно, только после модульного тестирования "следующая вещь в цепочке обработки", чтобы подтвердить, что она выполняет свою работу правильно). Поскольку API-интерфейсы являются интерфейсом к цепочке обработки, легко спутать модульное тестирование API с интеграционным тестированием всей этой цепочки обработки. Эффективная стратегия тестирования API предотвращает это. Чтобы продемонстрировать, как эффективно управлять тестированием API, я собираюсь рассмотреть стратегию модульного тестирования API (с использованием [Telerik Test Studio для API), чтобы продемонстрировать, как выглядит эффективная стратегия тестирования API в реальных условиях.

6. Apache Spark - популярная система с открытым исходным кодом для обработки больших объемов данных. Большие данные могут быть большими — возможно, петабайтами. Он также может быть сложным, объединяющим несколько схем и источников. Это также может быть высокоскоростной поток данных. Все эти типы данных выигрывают от обширной параллельной обработки различных частей данных в памяти, чтобы свести к минимуму циклические перемещения в постоянное хранилище. Apache Spark облегчает это, автоматически масштабируя обработку и память для эффективного анализа больших наборов данных. Приложения для Apache Spark включают пакетную обработку фильтрации данных, агрегирование данных и преобразование данных в пригодные для использования наборы данных. Он также может использовать модели машинного обучения и анализа для обработки огромных объемов данных для создания моделей, поиска тенденций и прогнозирования будущих сценариев. Это также полезно для обработки данных в режиме реального времени, которая быстро обрабатывает и анализирует поток данных. Многие другие области применения Apache Spark ограничены только вашим воображением.

6 Контрольные вопросы:

1. Постройте биграммную языковую модель на основе корпуса – «Вася любит мороженное. Лена любит малину. Вася любит Лену. Георгий ест мороженное. Лена рисует яблоко. Георгий любит Катю. Георгий любит смотреть, как Лена ест мороженное». Вычислите перплексию модели на предложении «Георгий любит малину».
2. Вычислите косинусную меру близости между документами «красный синий зеленый красный фисташковый» и «фисташковый алый лазоревый белый красный». В колеекции всего 10 000 документов. Документные частоты терминов: красный 100, синий 80, зеленый 75, фисташковый 10, алый 50, лазоревый 25, белый 200.
3. Есть коллекция документов: «Doc1: яблоко груша яблоко апельсин слива, Doc2: банан ананас яблоко ананас ананас, Doc3: лимон апельсин мандарин помело лайм, Doc4: яблоко дуриан папайя манго маракуйя» и запрос «Q: яблоко банан».Постройте смешанную языковую модель для поиска (смесь моделей по документу и коллекции; вес модели по документу \lambda=0.7λ=0.7) и ранжируйте документы по уменьшению вероятности соответствия документа запросу.
4. На рисунке 28 представлен веб-граф (веб-страницы и ссылки между ними). Вычислите значения PageRank для страниц с учетом вероятности телепортации 0.15 и упорядочите страницы по убыванию PageRank.
5. Есть оценка top5 результатов поиска по пяти запросам (Q1...Q5); +: релевантный документ, -: нерелевантный документ (рис. 5). Вычислите макроусредненную среднюю точность (mean average precision, MAP).
6. Какие вам известны инициативы/наборы данных для оценки методов морфологического анализа?
7. Опишите состав языковой модели на основе n-грамм: идея, реализация*, области применения.
8. Расскажите о подходах к оценке языковых моделей: перплексия* и "внешняя" оценка.
9. Каков механизм борьбы с разреженностью данных: сглаживание*, откат, интерполяция.
10. Опишите состав языковой модели на основе нейронных сетей.
11. Приведите формульное выражение законов Ципфа и Хипса, поясните их смысл.

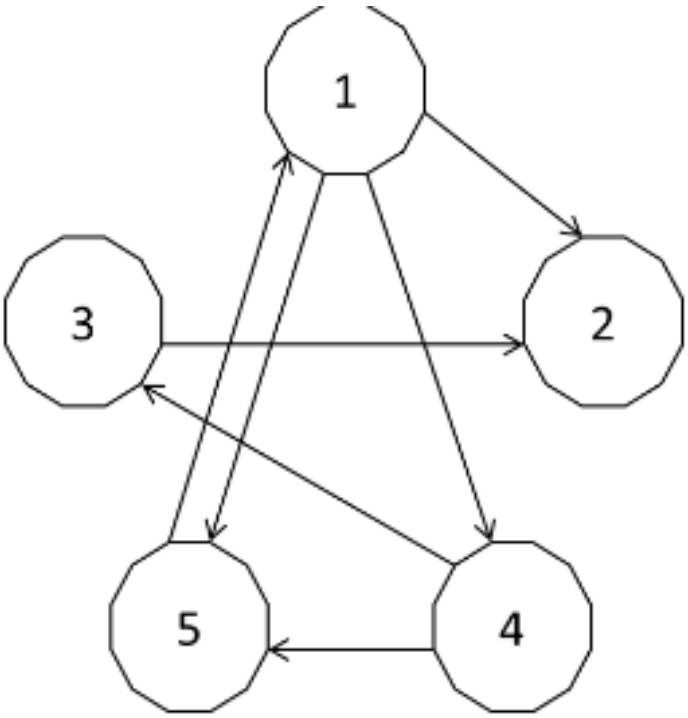

Рисунок 4 – Веб-граф

	Q1	Q2	Q3	Q4	Q5
1	+	-	-	-	-
2	-	+	-	-	-
3	-	+	-	+	+
4	+	-	+	-	+
5	-	-	+	-	-

Рисунок 5 – Оценка результатов поиска

Практическая работа №3 «Анализ тональности»

1 **Цель работы**: изучить способы определения тональности текстов.

2 **Порядок выполнения работы**:

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.

3 **Краткие теоретические сведения**

Многие люди во всем мире сейчас используют блоги, форумы и сайты социальных сетей, такие как Twitter и Facebook, чтобы поделиться своим мнением с остальной частью земного шара. Социальные сети стали одним из наиболее эффективных доступных средств коммуникации. В результате генерируется достаточное количество данных, называемых большими данными, и для эффективного и действенного анализа этих больших данных был введен анализ настроений. Для отрасли или организации стало исключительно важно понимать чувства пользователя. Анализ настроений, часто известный как анализ мнений, представляет собой метод определения того, является ли точка зрения автора или пользователя на тему положительной или отрицательной. Анализ настроений определяется как процесс получения значимой информации и семантики из текста с использованием естественных методов обработки и определения отношения автора, которое может быть положительным, отрицательным или нейтральным. Поскольку цель анализа настроений состоит в том, чтобы определить полярность и классифицировать высказанные мнения как положительные или отрицательные, диапазон классов dataset, участвующих в анализе настроений, не ограничивается только положительным или отрицательным; это может быть согласовано или не согласовано, хорошо или плохо. Это также может быть оценено количественно по 5-балльной шкале: категорически не согласен, не согласен, нейтрален, согласен или полностью согласен. Например, Ye и др. применили анализ настроений к отзывам о европейских и американских направлениях, отмеченных по шкале от 1 до 5. Они связали 1-звездочные или 2-звездочные отзывы с отрицательной полярностью и более чем 2-звездочные отзывы с положительной полярностью. Гребнер и др. создали специфичный для домена лексикон, состоящий из токенов с их значением настроения. Эти токены были собраны из отзывов клиентов в области туризма, чтобы классифицировать настроения по 5-звездочным рейтингам от ужасных до превосходных в области туризма. Кроме того, анализ настроений по тексту может быть выполнен на трех уровнях. Салинка применил алгоритмы машинного обучения к набору данных Yelp, который содержит обзоры поставщиков услуг в масштабе от 1 до

5. Анализ настроений можно разделить на три уровня, упомянутые в следующем разделе.

Анализ настроений возможен на трех уровнях: уровне предложения, уровне документа и уровне аспекта. При анализе настроений на уровне предложений или фраз документы или абзацы разбиваются на предложения, и определяется полярность каждого предложения. На уровне документа настроение определяется по всему документу или записи. Необходимость анализа настроений на уровне документа заключается в извлечении глобальных настроений из длинных текстов, содержащих избыточные локальные шаблоны и много шума. Наиболее сложным аспектом классификации настроений на уровне документа является учет связи между словами и фразами и полного контекста семантической информации для отражения состава документа. Это требует более глубокого понимания сложной внутренней структуры чувств и зависимых слов. На уровне аспекта определяется анализ настроений, мнение о конкретном аспекте или функции. Например, скорость процессора высока, но этот продукт имеет завышенную цену. Здесь скорость и стоимость - это два аспекта или точки зрения. Скорость упоминается в предложении, поэтому называется явным аспектом, тогда как стоимость является неявным аспектом. Анализ настроений на уровне аспектов немного сложнее, чем два других, поскольку трудно идентифицировать неявные признаки. Деви Шри Нандхини и Прадип предложили алгоритм для извлечения неявных аспектов из документов на основе частоты совпадения аспекта с индикатором признаков и использования связи между категоричными словами и явными аспектами. Ма и др. позаботился о двух проблемах, касающихся анализа на уровне аспектов: различные аспекты в одном предложении, имеющие разную полярность, и четкое положение контекста в высказанном предложении. Авторы создали двухэтапную модель на основе LSTM с механизмом внимания для решения этих проблем. Они предложили эту модель, основанную на предположении, что контекстные слова, близкие к аспекту, более актуальны и требуют большего внимания, чем более удаленные контекстные слова. На первом этапе модель использует несколько аспектов в предложении один за другим с помощью механизма позиционного внимания. Затем, во втором состоянии, он идентифицирует пары (аспект, предложение) в соответствии с положением аспекта и контекстом вокруг него и вычисляет полярность каждой команды одновременно.

Как указывалось ранее, анализ настроений и анализ эмоций часто используются исследователями как синонимы. Однако они отличаются несколькими способами. При анализе настроений первостепенной задачей является полярность, тогда как при обнаружении эмоций определяется эмоциональное или психологическое состояние или настроение. Анализ настроений исключительно субъективен, в то время как обнаружение эмоций более объективно и точно

Эмоции - неотъемлемая составляющая человеческой жизни. Эти эмоции влияют на принятие решений человеком и помогают нам лучше общаться с

миром. Обнаружение эмоций, также известное как распознавание эмоций, - это процесс идентификации различных чувств или эмоций человека (например, радости, печали или ярости). В течение последних нескольких лет исследователи усердно работали над автоматизацией распознавания эмоций. Однако некоторые физические нагрузки, такие как частота сердечных сокращений, дрожь в руках, потливость и высота голоса, также передают эмоциональное состояние человека, но распознавание эмоций по тексту довольно сложно. Кроме того, различные двусмысленности и новый сленг или терминология, появляющиеся с каждым днем, усложняют распознавание эмоций по тексту. Кроме того, обнаружение эмоций не ограничивается только определением основных психологических состояний (радость, грусть, гнев); вместо этого оно имеет тенденцию достигать 6- или 8-балльной шкалы в зависимости от модели эмоций.

В английском языке слово "эмоция" появилось в семнадцатом веке и произошло от французского слова "эмоция", означающего физическое расстройство. До девятнадцатого века страсть, аппетит и привязанности относились к категории психических состояний. В девятнадцатом веке слово "эмоция" считалось психологическим термином. В психологии сложные состояния чувств приводят к изменению мыслей, действий, поведения и личности, называемым эмоциями. В широком смысле психологические или эмоциональные модели подразделяются на две категории: пространственные и категориальные.

Размерная модель эмоций Эта модель представляет эмоции, основанные на трех параметрах: валентности, возбуждении и силе. Валентность означает полярность, а возбуждение означает, насколько захватывающим является чувство. Например, восхищение более волнующе, чем счастье. Власть или доминирование означает ограничение эмоций. Эти параметры определяют положение психологических состояний в 2-мерном пространстве, как показано на рисунке 1.

В категориальной модели эмоции определяются дискретно, такие как гнев, счастье, печаль и страх. В зависимости от конкретной категориальной модели эмоции подразделяются на четыре, шесть или восемь категорий.

В области обнаружения эмоций большинство исследователей приняли модель эмоций Экмана и Плутчика. Эмоциональные состояния, определяемые моделями, составляют набор меток, используемых для аннотирования предложений или документов. Батбаатар и др., Беккер и др., Джейн и др. приняли шесть основных эмоций Экмана. Сайлуназ и Альхаджж использовали модели Экмана для аннотирования твитов. Некоторые исследователи использовали индивидуальные модели эмоций, расширяя модель одним или двумя дополнительными состояниями. Робертс и др. использовали модель Экмана для аннотирования твитов состоянием "любовь".

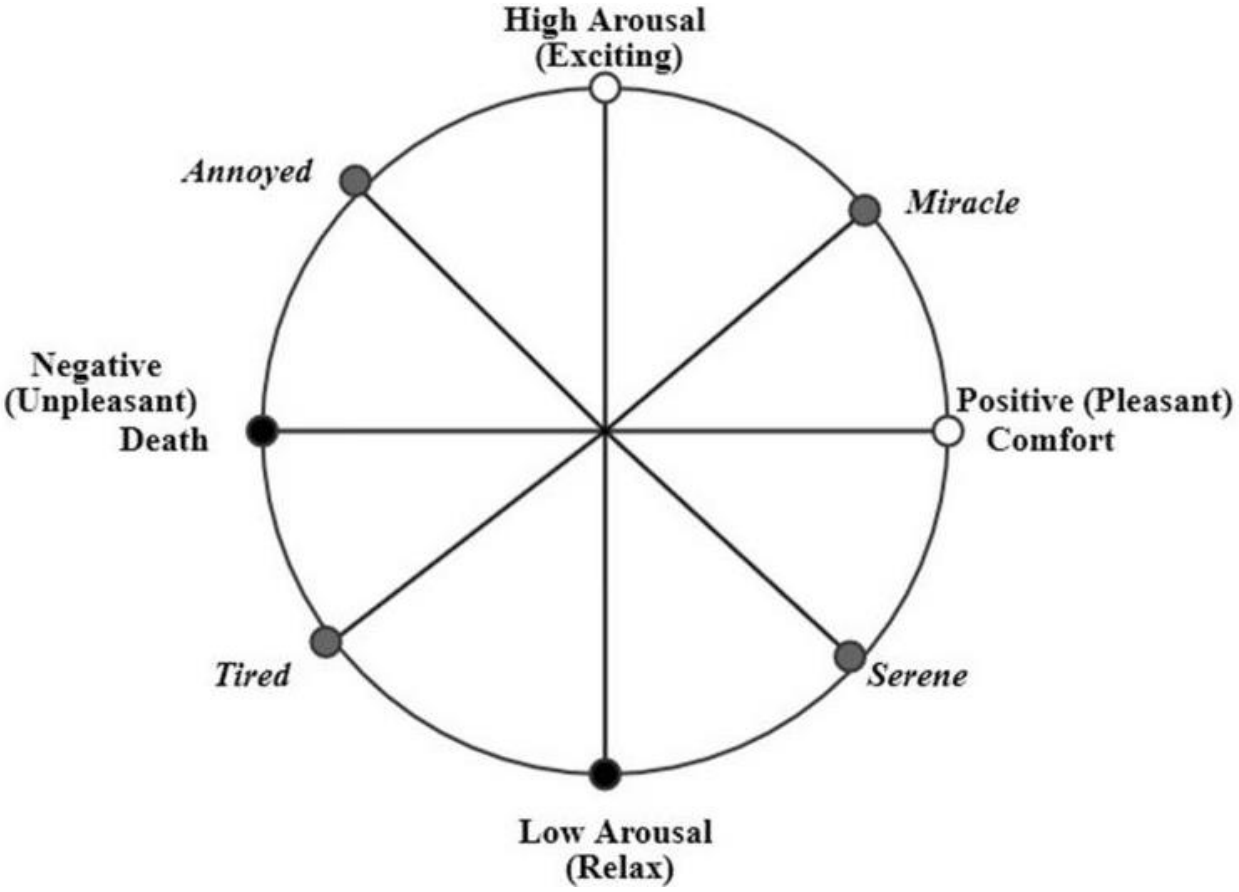

Рисунок 1 – Размерная модель эмоций

Ахмад и др. приняли колесо эмоций, смоделированное Плутчиком, для обозначения предложений на хинди девятью различными состояниями модели Плутчика, уменьшая, среди прочего, семантическую путаницу. Состояния модели Плутчика и Экмана также используются в различных лексиконах ручной работы, таких как WordNet-Аффект и словарно–эмоциональные лексиконы NRC . Лауберт и Парламис ссылались на модель Шейвера из-за ее трехуровневой иерархической структуры эмоций. Валентность или полярность представлена на первом уровне, за которым следует второй уровень, состоящий из пяти эмоций, а третий уровень показывает дискретные 24 эмоциональных состояния. Некоторые исследователи не ссылались ни на какую модель и классифицировали набор данных по трем основным чувствам: счастье, грусть или гнев.

В социальных сетях люди обычно легко делятся своими чувствами и эмоциями. В результате данные, полученные из сообщений, проверок, комментариев, замечаний и критических замечаний этих платформ социальных сетей, являются крайне неструктурированными, что затрудняет анализ настроений и эмоций для машин. В результате предварительная обработка является критическим этапом очистки данных, поскольку качество данных существенно влияет на многие подходы, которые следуют за предварительной обработкой. Организация набора данных требует предварительной обработки, включая маркировку, удаление стоп-слов, маркировку POS и т.д. Некоторые из

этих методов предварительной обработки могут привести к потере важной информации для анализа настроений и эмоций, что необходимо учитывать.

Токенизация - это процесс разбиения всего документа, абзаца или только одного предложения на фрагменты слов, называемые токенами. Например, рассмотрим предложение “это место такое красивое”, и после токенизации оно станет "этим", "местом", "таким" красивым.’ Крайне важно нормализовать текст для достижения единообразия данных путем преобразования текста в стандартную форму, исправления написания слов и т.д..

Необходимо удалить ненужные слова, такие как артикли и некоторые предлоги, которые не способствуют распознаванию эмоций и анализу настроений. Например, стоп-слова, такие как "есть", "в", "ан", "в", не имеют ничего общего с чувствами, поэтому их необходимо удалить, чтобы избежать ненужных вычислений. Пометка POS — это способ идентифицировать различные части речи в предложении. Этот шаг полезен для поиска различных аспектов предложения, которые обычно описываются существительными или именными фразами, в то время как чувства и эмоции передаются прилагательными.

Стемминг и лемматизация - два важнейших этапа предварительной обработки. В основе слова преобразуются в корневую форму путем усечения суффиксов. Например, термины "спорил" и "спорить" становятся "спорить". Этот процесс уменьшает нежелательное вычисление предложений. Лемматизация включает морфологический анализ для удаления флективных окончаний из лексемы, чтобы превратить ее в базовую лемму слова. Например, термин "пойманный" преобразуется в "улов". Симеонидис и др. изучили производительность четырех моделей машинного обучения с использованием комбинации и абляции различных методов предварительной обработки двух наборов данных, а именно SS-Tweet и SemEval. Авторы пришли к выводу, что удаление чисел и лемматизация повысили точность, тогда как удаление знаков препинания не повлияло на точность.

Машина понимает текст в терминах чисел. Процесс преобразования или отображения текста или слов в вещественные векторы называется векторизацией слов или встраиванием слов. Это метод извлечения объектов, при котором документ разбивается на предложения, которые затем разбиваются на слова; после этого строится карта объектов или матрица. В результирующей матрице каждая строка представляет предложение или документ, в то время как каждый столбец объектов представляет слово в словаре, а значения, присутствующие в ячейках карты объектов, обычно обозначают количество слов в предложении или документе. Для извлечения признаков одним из наиболее простых методов является "Набор слов" (BOW), в котором определяется вектор подсчета фиксированной длины, где каждая запись соответствует слову в заранее определенном словаре слов. Слову в предложении присваивается значение 0, если оно отсутствует в заранее определенном словаре, в противном случае значение больше или равно 1 в зависимости от того, сколько раз оно встречается в предложении. Вот почему

длина вектора всегда равна словам, присутствующим в словаре. Преимуществом этого метода является его простота реализации, но он имеет существенные недостатки, поскольку он приводит к разреженной матрице, теряет порядок слов в предложении и не отражает смысла предложения. Например, для представления текста “вам нравится читать” из заранее определенного словаря, я надеюсь, что вы, наслаждаетесь чтением, было бы (0,0,1,1,1,1). Однако эти представления могут быть улучшены путем предварительной обработки текста и использования n-граммы, TF-IDF.

Метод N-грамм является отличным вариантом для определения порядка слов в векторном представлении предложений. В векторном представлении n-граммы текст представлен как совокупность уникальных групп n-граммных средств из n смежных терминов или слов. Значением n может быть любое натуральное число. Например, рассмотрим предложение “учить - значит прикасаться к жизни навсегда”, и n = 3, называемое триграммой, будет генерировать "учить", "учить", "прикасаться", "прикасаться", "прикасаться к жизни", "жизнь навсегда". Таким образом, порядок предложения может быть сохранен. Функции N-граммов работают лучше, чем подход BOW, поскольку они охватывают синтаксические шаблоны, включая важную информацию. Однако, хотя n-грамм сохраняет порядок слов, он обладает высокой размерностью и разреженностью данных.

Термин частота - обратная частота документа, обычно сокращенная как TFIDF, является еще одним методом, обычно используемым для извлечения признаков. Этот метод представляет текст в матричном виде, где каждое число количественно определяет, сколько информации несут эти термины в данном документе. Он построен на предпосылке, что редкие термины содержат много информации в текстовом документе. Частота термина - это количество раз, когда слово w встречается в документе, деленное на общее количество слов W в документе, а IDF - это журнал (общее количество документов (N), деленное на общее количество документов, в которых появляется слово w (n)). Ахуджа и др. внедрили шесть методов предварительной обработки и сравнили два метода извлечения признаков, чтобы определить наилучший подход. Они применили шесть алгоритмов машинного обучения и использовали n-граммы с n = 2 и TF-IDF для извлечения признаков из набора данных SS-tweet и пришли к выводу, что TF-IDF обеспечивает лучшую производительность по сравнению с n-граммами.

Доступность огромных объемов данных позволяет сети глубокого обучения находить хорошие векторные представления. Извлечение признаков с помощью встраивания слов на основе нейронных сетей более информативно. При встраивании слов на основе нейронной сети слова с одинаковой семантикой или связанные друг с другом представлены похожими векторами. Это более популярно в предсказании слов, так как сохраняет семантику слов. Исследовательская группа Google, возглавляемая Томасом Миколовым, разработала модель Word2Vec для встраивания слов. С помощью Word2Vec для

машины можно понять, что векторное представление “королева” + “женщина” + “мужчина” будет таким же, как векторное представление “король”.

Другие примеры моделей встраивания слов на основе глубокого обучения включают Glove, разработанную исследователями из Стэнфордского университета, и FastText, представленный Facebook. Векторы перчаток обучаются быстрее, чем Word2vec. Векторы FastText обладают большей точностью по сравнению с векторами Word2Vec по нескольким различным показателям. Янг и др. доказали, что выбор подходящего встраивания слов на основе нейронных сетей может привести к значительным улучшениям даже в случае отсутствия словарного запаса (OOV). Авторы сравнили различные встраивания слов, обученные с использованием Twitter и Википедии в качестве корпусов, с встраиванием слов TF-IDF.

Человек может выражать свои эмоции в любой форме, такой как выражение лица, жесты, речь и текст. Обнаружение эмоций в тексте — это проблема классификации на основе контента. Далее произведём разбор задачи обнаружения эмоций в тексте с помощью машинного обучения с использованием Python.

В машинном обучении обнаружение текстовых эмоций является проблемой классификации на основе контента, которая является задачей обработки естественного языка. Распознавание эмоций человека - сложная задача, но распознавание эмоций с помощью текста, написанного человеком, еще сложнее, поскольку человек может выражать свои эмоции в любой форме.

Обычно эмоции выражаются в виде радости, печали, гнева, удивления, ненависти, страха и т.д. Распознавание этого типа эмоций по тексту, написанному человеком, играет важную роль в таких приложениях, как чат-боты, форум поддержки клиентов, отзывы клиентов и т.д.

Для определения эмоций по тексту выполняются несколько шагов, которые начнутся с подготовки данных. Затем следующим шагом будет токенизация, при которой текстовые данные будут преобразованы в токены, и из этих токенов мы должны идентифицировать эмоциональные слова.

Эти эмоциональные слова будут ключевыми для классификации эмоций текста. Далее мы сформулируем эту задачу таким образом, чтобы в качестве входных данных был взят текст, а в качестве выходных данных был сгенерирована текстовая фраза, представляющая эмоции в этом тексте.

Сначала мы импортируем необходимые пакеты. Если ранее не были установлены эти пакеты, можно установить их с помощью pip.

```
import pandas as pd
import numpy as np
import re
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.utils import to_categorical
from keras.models import load_model
```

```
import urllib.request
import zipfile
import os
from keras.models import Sequential
from keras.layers import Embedding,Bidirectional,LSTM,GRU,Dense
import nltk
from nltk.tokenize import word_tokenize
import warnings
import tensorflow as tf
nltk.download('punkt')
warnings.filterwarnings('ignore')
```

Теперь, когда пакеты импортированы, необходимо извлечь предложения и соответствующие им эмоции и вставить их в фреймы данных обучения, тестирования и проверки соответственно.

Поскольку файлы находятся в формате .txt, нужно использовать приведенный ниже код, чтобы поместить их в обучающие и тестовые (включая проверку) фреймы данных.

```
f=open('train.txt','r')
x_train=[]
y_train=[]
for i in f:
l=i.split(';')
y_train.append(l[1].strip())
x_train.append(l[0])
f=open('test.txt','r')
x_test=[]
y_test=[]
for i in f:
l=i.split(';')
y_test.append(l[1].strip())
x_test.append(l[0])
f=open('val.txt','r')
for i in f:
l=i.split(';')
y_test.append(l[1].strip())
x_test.append(l[0])
data_train=pd.DataFrame({'Text':x_train,'Emotion':y_train})
data_test=pd.DataFrame({'Text':x_test,'Emotion':y_test})
data=data_train.append(data_test,ignore_index=True)
```

Теперь, когда предложения вставлены, нужно их очистить. По сути, необходимо удаить все предлоги, артикли, знаки препинания, стоп-слова,

оставляя в предложениях только важные слова. Здесь удаляемые слова действуют как шум, поэтому важно устранить их, чтобы получить желаемый результат, т.е. высокую точность тестирования.

```
def clean_text(data):
data=re.sub(r"(#[\d\w\.]+)", '', data)
data=re.sub(r"(@[\d\w\.]+)", '', data)
data=word_tokenize(data)
return data
texts=[' '.join(clean_text(text)) for text in data.Text]
texts_train=[' '.join(clean_text(text)) for text in x_train]
texts_test=[' '.join(clean_text(text)) for text in x_test]
```

Токенизация является важным процессом в анализе NLP (Обработка естественного языка). Он маркирует каждое предложение, извлекает каждое уникальное слово и создает словарь, в котором каждому уникальному слову присваивается индекс.

```
tokenizer=Tokenizer()
tokenizer.fit_on_texts(texts)
sequence_train=tokenizer.texts_to_sequences(texts_train)
sequence_test=tokenizer.texts_to_sequences(texts_test)
index_of_words=tokenizer.word_index
vocab_size=len(index_of_words)+1
```

Полученный набор данных содержит шесть уникальных результатов или эмоций, а именно: гнев, печаль, страх, радость, удивление и любовь.

Следовательно, количество классов равно шести. Кроме того, здесь использовано 300 встроенных измерений, а максимальной длине последовательности присваивается значение 500.

Когда заполняются последовательности обучения и тестирования, крайне важно, чтобы их параметр "maxlen" имел одинаковое значение, если это не так, будет показана ошибка.

На более поздних стадиях каждой эмоции присваивается категориальное значение (0-5). Именно по этой причине используется словарь "кодирование" и функция "to_categorical". Когда происходит попытка получить результат, сопоставляются категориальное значение с эмоцией.

```
num_classes=6
embed_num_dims=300
max_seq_len=500
class_names=['anger','sadness','fear','joy','surprise','love']X_train_pad=pad_seq
uences(sequence_train,maxlen=max_seq_len)
X_test_pad=pad_sequences(sequence_test,maxlen=max_seq_len)encoding={'a
nger':0,'sadness':1,'fear':2,'joy':3,'surprise':4,'love':5}
```

```
y_train=[encoding[x] for x in data_train.Emotion]
y_test=[encoding[x] for x in data_test.Emotion]
y_train=to_categorical(y_train)
y_test=to_categorical(y_test)
```

Для создания этой модели необходимо использовать версию 1M (1 миллион векторов слов, обученных в Википедии) предварительно обученных векторов слов.

Существуют обученные векторы слов, которые можно использовать для этой цели. Использование этих векторов слов помогает обучать модель более эффективно и тщательно, что, в свою очередь, обеспечивает более высокую точность.

```
def create_embedding_matrix(filepath,word_index,embedding_dim):
   vocab_size=len(word_index)+1
   embedding_matrix=np.zeros((vocab_size,embedding_dim))
   with open(filepath) as f:
      for line in f:
         word,*vector=line.split()
         if word in word_index:
            idx=word_index[word]
            embedding_matrix[idx] =
np.array(vector,dtype=np.float32)[:embedding_dim]
   return embedding_matrixfname='embeddings/wiki-news-300d-1M.vec'
embedd_matrix=create_embedding_matrix(fname,index_of_words,embed_nu
m_dims)
```

Теперь необходимо создать архитектуру, которая будет использоваться для обучения модели. Для этой цели сначала создается слой встраивания, для которого веса получены из файла word vectors.

Также необходимо добавть двунаправленный слой, функциями которого являются:

- gru_output_size = 128
- dropout = 0.2
- recurrent_dropout = 0,2

Наконец, добавляется плотный слой с активацией "softmax".

В качестве оптимизатора используется оптимизатор Адама, а потери рассчитываются с использованием ‘categorical_crossentropy’. ‘model.summary()’ можно использовать для просмотра объектов, типа слоя, формы вывода и количества параметров в модели.

```
embedd_layer=Embedding(vocab_size,embed_num_dims,input_length=max_s
eq_len,weights=[embedd_matrix],trainable=False)
gru_output_size=128
```

```
bidirectional=True
model=Sequential()
model.add(embedd_layer)
model.add(Bidirectional(GRU(units=gru_output_size,dropout=0.2,recurrent_dr
opout=0.2)))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['acc
uracy'])
```

Наконец, можно обучить модель, используя обучающий набор, и одновременно проверить точность, поскольку для показателя модели установлено значение "точность". Здесь размер пакета принимается равным 128, а количество эпох равно 8.

Размер пакета и количество эпох могут быть изменены. Количество эпох не должно быть слишком большим, чтобы избежать переобучения. Размер пакета также может варьироваться, во многих случаях большие размеры пакета дают лучшие результаты, но они также занимают много памяти и, следовательно, невозможны для выполнения во многих системах.

```
batch_size=128
epochs=8
hist=model.fit(X_train_pad,y_train,batch_size=batch_size,epochs=epochs,vali
dation_data=(X_test_pad,y_test))
```

После завершения 8 эпох необходимо протестировать модель. В ходе разработки было произведено преобразование эмоции в категориальные или числовые значения, поэтому для получения точной эмоции необходимо сопоставить категориальное значение с фактической эмоцией.

```
message=['I am sad.']
seq=tokenizer.texts_to_sequences(message)
padded=pad_sequences(seq,maxlen=max_seq_len)
pred=model.predict(padded)
print('Message:'+str(message))
print('Emotion:',class_names[np.argmax(pred)])
```

Для приведенного выше предложения результат - "печаль", что, вероятно, правильно. Путем объединения приведенных выше фрагментов кода можно получить всю программу целиком. Для выполнения этой программы можно использовать Google Colab, поскольку в ней есть встроенный графический процессор, который полезен при обучении моделям машинного обучения, если в вашей системе уже нет графического процессора. Полезный лакомый кусочек - сохраните текстовую модель для дальнейшего использования (вы никогда не знаете, когда она вам понадобится).

tf.keras.models.save_model(model,'textmodel',overwrite=True,include_optimizer=True,save_format=None,signatures=None,options=None)

Используя эту текстовую модель, можно создать тест, который определяет эмоции пользователя, или выполнить анализ настроений твитов или сообщений Reddit, возможности безграничны.

Для каждого пакета отделяем текст (text) от меток (labels) и передаем их в оптимизатор nlp.update(). Это фактически запускает обучение.

Параметр dropout сообщает nlp.update(), какую часть обучающих данных в этом пакете нужно пропустить. Это делается для того, чтобы модели было сложнее переобучиться – запомнить обучающие данные без создания обобщающей модели.

Поскольку мы будем выполнять множество оценок на большом количестве вычислений, имеет смысл написать отдельную функцию evaluate_model(). В этой функции мы будем классифицировать тексты из валидационного набора данных на недообученной модели и сравнивать результаты модели с метками исходных данных.

Используя эту информацию, мы вычислим следующие метрики:

- Истинно положительные (true positives, TP) – число отзывов, которые модель правильно предсказала как положительные.
- Ложноположительные (false positives, FP) – число отзывов, которые модель неверно предсказала как положительные, хотя на самом деле они были негативными.
- Истинно отрицательные (true negatives, TN) – число отзывов, которые модель правильно предсказала как негативные.
- Ложноотрицательные (false negatives, FN) – число отзывов, которые модель неверно предсказала как негативные, хотя на самом деле они были положительными.

Поскольку наша модель для каждой метки возвращает оценку от 0 до 1, мы определяем положительный или отрицательный результат на основе этой оценки. На основе четырех описанных статистических данных мы вычисляем две метрики: точность и полноту. Эти метрики являются показателями эффективности модели классификации:

- Точность (precision) – отношение истинно положительных результатов ко всем элементам, отмеченным моделью как положительные (истинные и ложные срабатывания). Точность 1.0 означает, что каждый отзыв, отмеченный нашей моделью как положительный, действительно относится к положительному классу.
- Полнота (recall) – это отношение истинно положительных отзывов ко всем фактическим положительным отзывам, то есть количество истинно положительных отзывов, деленных на суммарное количество истинно положительных и ложноотрицательных отзывов.

$$precision = \frac{TP}{TP + FP}, recall = \frac{TP}{TP + FN}.$$

Ещё более популярной метрикой является F1-мера – среднее гармоническое точности и полноты. Максимизация F1-меры приводит к одновременной максимизации этих двух критериев:

$$F_1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

В evaluate_model() необходимо передать токенизатор, textcat и тестовый набор данных.

```
def evaluate_model(tokenizer, textcat, test_data: list) -> dict:
    reviews, labels = zip(*test_data)
    reviews = (tokenizer(review) for review in reviews)
    # Указываем TP как малое число, чтобы в знаменателе
    # не оказался 0
    TP, FP, TN, FN = 1e-8, 0, 0, 0
    for i, review in enumerate(textcat.pipe(reviews)):
        true_label = labels[i]['cats']
        score_pos = review.cats['pos']
        if true_label['pos']:
            if score_pos >= 0.5:
                TP += 1
            else:
                FN += 1
        else:
            if score_pos >= 0.5:
                FP += 1
            else:
                TN += 1
    precision = TP / (TP + FP)
    recall = TP / (TP + FN)
    f_score = 2 * precision * recall / (precision + recall)
    return {"precision": precision, "recall": recall, "f-score": f_score}
```

В этой функции мы разделяем обзоры и их метки, затем используем выражение-генератор для токенизации каждого из обзоров, подготавливая их для передачи в textcat. Выражение-генератор позволяет перебирать токенизированные обзоры, не сохраняя каждый из них в памяти.

Затем мы используем score и true_label для определения ложных и истинных срабатываний модели, которые далее подставляем для расчета точности, полноты и F-меры.

Вызовем evaluate_model() из описанной ранее функции train_model():

```
import os
import random
import spacy
from spacy.util import minibatch, compounding

def train_model(
    training_data: list,
    test_data: list,
    iterations: int = 20) -> None:
    # Строим конвейер
    nlp = spacy.load("en_core_web_sm")
    if "textcat" not in nlp.pipe_names:
        textcat = nlp.create_pipe(
            "textcat", config={"architecture": "simple_cnn"}
        )
        nlp.add_pipe(textcat, last=True)
    else:
        textcat = nlp.get_pipe("textcat")

    textcat.add_label("pos")
    textcat.add_label("neg")

    # Обучаем только textcat
    training_excluded_pipes = [
        pipe for pipe in nlp.pipe_names if pipe != "textcat"
    ]
    with nlp.disable_pipes(training_excluded_pipes):
        optimizer = nlp.begin_training()
        # Training loop
        print("Начинаем обучение")
        print("Loss\t\tPrec.\tRec.\tF-score")          #
        batch_sizes = compounding(
            4.0, 32.0, 1.001
        )  # Генератор бесконечной последовательности входных чисел
        for i in range(iterations):
            loss = {}
            random.shuffle(training_data)
            batches = minibatch(training_data, size=batch_sizes)
            for batch in batches:
                text, labels = zip(*batch)
                nlp.update(
                    text,
```

```
                labels,
                drop=0.2,
                sgd=optimizer,
                losses=loss
            )
        with textcat.model.use_params(optimizer.averages):
            evaluation_results = evaluate_model(   #
                tokenizer=nlp.tokenizer,          #
                textcat=textcat,                  #
                test_data=test_data               #
            )                                     #
            print(f"{loss['textcat']:9.6f}\t\
{evaluation_results['precision']:.3f}\t\
{evaluation_results['recall']:.3f}\t\
{evaluation_results['f-score']:.3f}")

    # Сохраняем модель                            #
    with nlp.use_params(optimizer.averages):      #
        nlp.to_disk("model_artifacts")            #
```

4 Задания для выполнения работы

– определить тональность текста по шкале от 1 (негативный) до 10 (позитивный).

– ответить на контрольные вопросы

Ввод: тексты, разделенные переводом строки (\n).

Вывод: для каждого текста из входных данных вывести тональную оценку от 1 до 10. Разделитель между выводами для разных текстов -- перевод строки (\n).

В качестве оценки используется квадратный корень из среднеквадратической ошибки (Root Mean Squared Error).

Sample Input:

```
Очень хорошо прописаны характеры. Занусси словно под микроскопом
разглядывает человека, препарирует ситуацию, разбирает случай на
детали, устраивает качели игр разума. Жанр не мой, но досмотрела
— захватывает.
на мой взгляд самая неудачная книга у этого автора...
```

Sample Output:

```
8
5
```

Методика выполнения

1. Для выполнения практического задания создать программу на языке Python(рисунок 2), которая производит анализ тональности предложений и выводит оценку тональности текста от 1 до 10.

2. Данный язык необходимо выбрать, так как его инструментарий содержит все необходимые инструменты для автоматического реферирования документов. В частности, использовать пакеты sklearn, numpy, pandas, nltk.

3. Исходный текст записать в файл for_sentient.txt

["А что у вас , ребята ?\nПереиздание легендарного « Томека » будит в душе чувства , которые , возможно , не стоило бы тревожить понапрасну .\n\nПри виде нового , оформленного под путевой дневник издания легендарного « Томека » сердца бывших мальчишек наверняка забьются чаще .\nПо крайней мере тех , кто помнит фантастические приключения Томека Вильмовского и его старших друзей в Австралии , Африке , Сибири , Америках и много где ещё .\nПомнит ловко пойманных или подстреленных Томеком редких зверей , помнит снежного человека , пигмеев с отравленными стрелами , охотников за головами и агентов царской полиции .\nИ , пожалуй , сердце у них забьётся не вовсе напрасно .\nКонечно , книжки о приключениях Томека с 1970-х годов переиздавались у нас множество раз , в том числе и в последнее время ; их общий тираж — несколько миллионов .\nТак что никто вроде не мешает читать их хоть на бумаге , хоть в Сети .\nОднако дело в том , что на этот раз за издание серии романов Альфреда Шклярского принялось уважаемое издательство « Розовый жираф » , а это обычно служит знаком правильной селекции книжек .\n\nПока вышло два первых тома , и вот что о них следует знать .\nОформление , как уже было сказано , отличное , к тому же обе книжки заново проиллюстрированы .\nТомек на новых картинках похож на юного Лео Ди Каприо , но это выглядит даже органичнее , чем старые , классические рисунки , на которых мальчика изображали в виде карлика - ковбоя .\nЧто касается собственно текста , то , прежде чем давать его ребёнку , стоит хорошенько подумать .]

```
texts = []
for line in open('./data/for_sentiment.txt', 'r', encoding="utf-8"):
    texts.append(line.strip())


transtext = cv.transform(texts)

result = []
for rev in transtext:
    #print(final_model.predict(rev))
    result.append(final_model.predict(rev)[0])

print(result)

with open('./Data/sentiment_result.txt', 'w') as f:
    for item in result:
        f.write(item.astype(str) + '\n')
```

```
[10, 10, 9, 8, 8, 9, 8, 10, 5, 10, 10, 1, 8, 10, 9, 2, 9, 10, 9, 8, 8, 10, 10, 8, 10, 4, 9, 9, 9, 6, 10, 6, 4, 10, 8, 10, 10,
2, 1, 5, 10, 5, 5, 8, 9, 9, 10, 7, 8, 8, 4, 10, 8, 6, 6, 10, 5, 7, 9, 9, 9, 7, 6, 6, 9, 10, 8, 8, 5, 10, 8, 10, 9, 10, 8, 8, 1
0, 6, 8, 8, 10, 8, 8, 4, 9, 9, 8, 5, 10, 9, 10, 8, 1, 5, 7, 9, 9, 4, 9, 9, 9, 8, 8, 10, 6, 5, 10, 10, 9, 10, 8, 9, 10, 5, 10, 1
0, 9, 4, 7, 8, 9, 6, 10, 10, 7, 10, 10, 6, 10, 10, 10, 8, 5, 9, 9, 10, 5, 10, 10, 5, 8, 10, 9, 10, 10, 7, 10, 4, 9, 5, 7, 7, 6,
9, 9, 6, 10, 10, 7, 10, 8, 5, 9, 8, 10, 9, 9, 8, 9, 9, 9, 10, 9, 9, 10, 10, 9, 7, 8, 7, 9, 9, 9, 8, 9, 5, 10, 10, 8, 9, 4, 8, 1
0, 9, 8, 10, 1, 10, 5, 9, 9, 10, 10, 8, 10, 9, 10, 9, 4, 6, 10, 10, 10, 6, 8, 9, 6, 6, 10, 2, 10, 10, 7, 10, 10, 6, 9, 5, 10, 1
0, 10, 9, 10, 5, 9, 10, 3, 10, 10, 9, 9, 5, 9, 5, 9, 10, 9, 9, 10, 10, 5, 9, 9, 7, 9, 7, 10, 8, 10, 8, 8, 9, 10, 7, 9, 9, 10,
5, 10, 10, 10, 7, 8, 9, 9, 7, 9, 10, 6, 10, 9, 10, 2, 8, 4, 10, 9, 8, 9, 9, 9, 7, 10, 10, 5, 5, 8, 5, 10, 8, 10, 9, 9, 9, 10,
9, 5, 9, 10, 5, 10, 8, 8, 5, 9, 7, 6, 10, 7, 9, 10, 9, 8, 7, 9, 9, 3, 7, 5, 8, 5, 6, 8, 9, 6, 8, 6, 7, 6, 7, 9, 9, 3, 9, 7, 9,
10, 5, 9, 6, 10, 10, 10, 10, 10, 10, 5, 7, 10, 5, 8, 10, 8, 9, 10, 9, 10, 10, 7, 9, 10, 9, 9, 9, 10, 9, 10, 10, 8, 9, 10, 5, 4,
1, 10, 9, 5, 9, 10, 8, 10, 5, 9, 6, 8, 9, 9, 10, 10, 10, 9, 8, 8, 7, 9, 7, 7, 9, 10, 8, 9, 8, 6, 10, 10, 5, 10, 7, 10, 9, 5, 9,
5, 9, 9, 8, 9, 7, 10, 10, 10, 5, 8, 9, 9, 8, 8, 10, 10, 8, 8, 9, 9, 7, 7, 6, 10, 10, 10, 8, 8, 9, 9, 9, 9, 9, 6, 10, 8, 10, 4,
5, 7, 10, 8, 8, 7, 3, 9, 10, 10, 9, 8, 8, 10, 6, 9, 10, 8, 6, 9, 2, 10, 8, 10, 10, 10, 8, 10, 7, 10, 3, 9, 10, 9, 7, 8, 10, 8,
7, 10, 9, 10, 8, 7, 9, 9, 9, 10, 7, 8, 10, 9, 7, 7, 9, 10, 9, 9, 3, 9, 10, 9, 8, 8, 9, 7, 3, 4, 5, 3, 7, 9, 9, 10, 9, 7, 10, 7,
10, 10, 8, 10, 8, 8, 9, 9, 9, 5, 10, 10, 6, 9, 8, 6, 7, 5, 9, 8, 9, 9, 7, 7, 8, 10, 4, 6, 10, 9, 5, 9, 5, 9, 10, 10, 7, 6, 6, 1
0, 10, 7, 8, 10, 10, 8, 9, 5, 10, 9, 10, 10, 10, 10, 9, 4, 8, 7, 4, 10, 9, 9, 8, 10, 10, 10, 9, 7, 8, 10, 9, 8, 10, 8, 10, 9,
6, 10, 5, 8, 8, 7, 9, 10, 6, 9, 10, 10, 9, 9, 8, 10, 9, 10, 7, 8, 8, 4, 10, 8, 2, 10, 10, 4, 5, 8, 5, 9, 5, 10, 7, 5, 2, 8, 9,
7, 8, 9, 7, 7, 10, 9, 5, 8, 9, 5, 9, 6, 3, 4, 5, 8, 9, 9, 8, 9, 10, 10, 9, 10, 4, 8, 9, 9, 7, 9, 8, 8, 8, 4, 9, 9, 9, 9, 9, 8,
```

Рисунок 2 – Алгоритм обработки и выходные данные

4. В результате проверить готовое решение (рисунок 3).

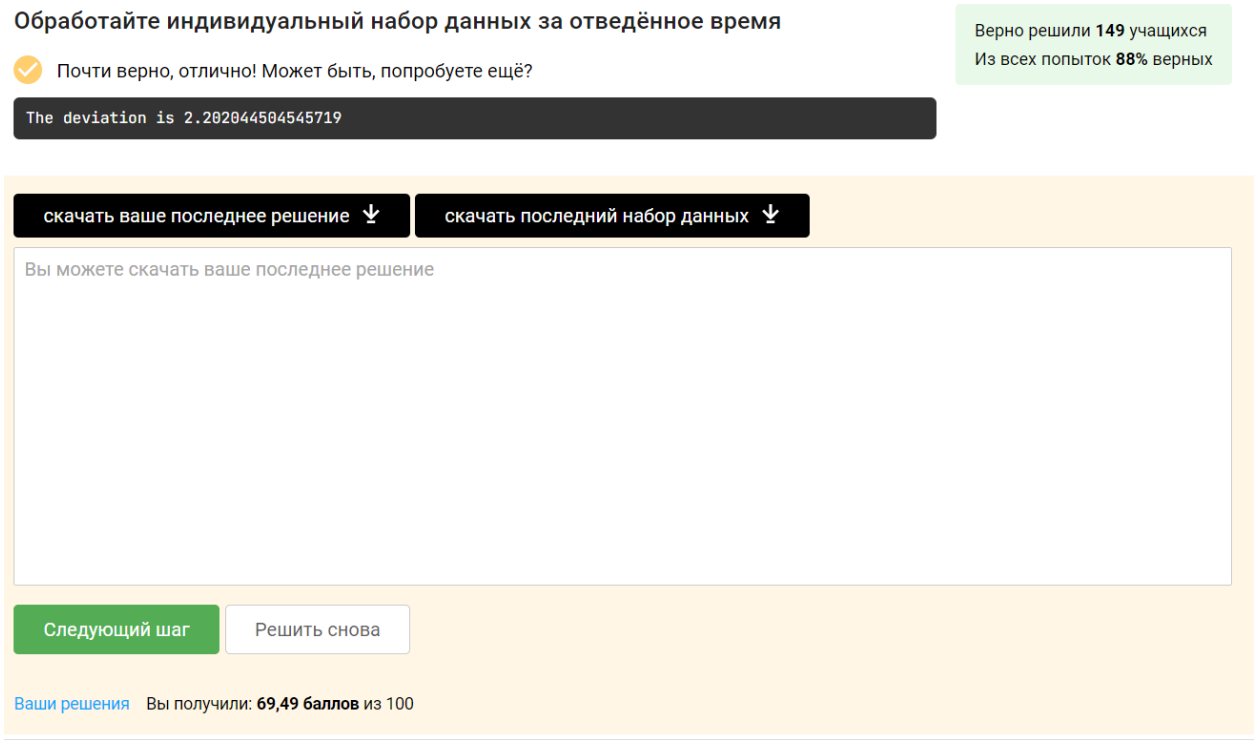

Рисунок 3 – Результаты проверки полученных результатов

5 Варианты заданий

1. Если у разработчика в его проекте есть модуль, который подключается к FTP-серверу для загрузки некоторых файлов, то разработчик напишет метод подключения к FTP через URL-адрес FTP и использует учетные данные, такие как имя пользователя и пароль для успешного подключения. Если разработчик использует эти учетные данные в коде и жестко закодирует их в своем файле кода и развернет приложение, оно может работать нормально, работа выполнена! Теперь представьте, что через два месяца пароль для этого FTP-сайта изменен, и разработчик должен снова обновить этот пароль в вашем приложении, чтобы убедиться, что он не нарушает существующую функциональность FTP-соединения. Теперь в этом сценарии разработчик снова изменит код и повторно развернет его. Здесь разработчик должен для небольшого изменения конфигурации использовать последнюю версию кода, внести необходимые изменения, убедиться, что больше ничего не сломается, повторно развернуть приложение и затем протестировать его. И это может быть повторяющейся задачей через 2-3 месяца, когда какая-то конфигурация снова изменится. Что делать, если в приложении есть файл конфигурации, в котором есть пары ключ-значение “Пользователь”: “<имя пользователя>”, “Пароль”: “пароль”, и всякий раз, когда требуется какое-либо изменение, затрагивается только этот файл, и конфигурации обновляются, а не копаются в фактическом коде.
2. Исходя из опыта работы с .NET, мне было немного сложно понять, как разработчик может иметь файл конфигурации в приложении Python, который можно использовать для чтения значений настроек, и при этом не нужно даже прикасаться к коду для обновления или сохранения настроек. Конфигурационные файлы используются для хранения пар ключ-значение или некоторой настраиваемой информации, которая может быть прочитана или доступна в коде и в определенный момент времени. Если эта конфигурация изменится, разработчики могут просто изменить конфигурацию в этом файле конфигурации и не беспокоиться об изменении кода, так как это может потребовать повторной компиляции кода и его развертывания. Не только это, но и использование конфигурационных файлов делает ваши настройки и код более удобными для повторного использования, а также позволяет хранить информацию о настройках централизованно и отдельно. Конечно, конфиденциальная информация, такая как пароли, секреты и сертификаты, должна храниться в большей безопасности и может находиться в облачных хранилищах. Но основные настройки, используемые в приложении, могут быть частью файла конфигурации.
3. Графики знаний существуют уже почти полвека – этот термин был впервые введен в обиход в 1972 году! Долгое время они просто томились в академическом мире, пока Google не анонсировала свой график знаний в 2012 году. С тех пор графики знаний эволюционировали довольно резко, и теперь

пути назад нет.За последние 10 лет наблюдается стремительный рост машинного обучения и искусственного интеллекта . Благодаря своей способности внедрять интеллектуальные данные в данные и добавлять контекст, графики знаний используются для того, чтобы сделать МО и ИИ более надежными, заслуживающими доверия и объяснимыми. Вполне естественно, что эти две технологии сближаются, стимулируя рост и необходимость графиков знаний. Графики знаний преобразуют интеллектуальные данные в данные с помощью контекста и взаимосвязей. Графики знаний организуют и предоставляют единое место для поиска соответствующих данных и – что не менее важно – для интерпретации и принятия мер в соответствии с тем, что важно. Связанные данные, обогащенные смыслом, позволяют лучше отвечать на сложные запросы и находить наиболее эффективный путь к бесценной информации.

4. Инкапсуляция — это концепция объектно-ориентированного программирования, которая охватывает детали реализации класса. Почему бы не раскрыть детали реализации? Одна из хороших практик программирования заключается в том, что клиентский код должен иметь доступ только к открытым интерфейсам класса. Пока интерфейс остается прежним, клиентский код не нуждается в изменении в случае изменения реализации. Инкапсуляция также снижает сложность и упрощает процессы отладки. Кроме того, инкапсулированный класс помогает защитить данные и предотвратить неправильное использование, поскольку клиентский код не может получить доступ к защищенной части класса. В дополнение к деталям реализации, элементы данных, как правило, также являются участками кода, которые должны быть защищены. Однако может оказаться целесообразным предоставить общедоступный интерфейс, позволяющий клиентскому коду получать доступ к элементам данных в контексте класса. Этот тип интерфейса обычно называется функцией доступа. Функции доступа обычно бывают двух видов: геттер и сеттер. Получатель — это метод, который вызывается, когда мы обращаемся к элементу данных для чтения. В отличие от геттера, сеттер — это метод, вызываемый для изменения элемента данных.

5. "Тестирование кода" обычно организуется в виде пути, который начинается с модульного тестирования, продолжается интеграцией/сквозным тестированием и заканчивается приемочным тестированием пользователей. В начале этого пути модульное тестирование выполняет две вещи — оно подтверждает, что "тестируемый код" работет правильно при подаче своих входных данных. После этого вы переходите к интеграционному тестированию, где подтверждаете, что "передача" работает правильно (предположительно, только после модульного тестирования "следующая вещь в цепочке обработки", чтобы подтвердить, что она выполняет свою работу правильно). Поскольку API-интерфейсы являются интерфейсом к цепочке обработки, легко спутать модульное тестирование API с интеграционным тестированием всей этой цепочки обработки. Эффективная стратегия тестирования API предотвращает это. Чтобы продемонстрировать, как эффективно управлять

тестированием API, я собираюсь рассмотреть стратегию модульного тестирования API (с использованием [Telerik Test Studio для API), чтобы продемонстрировать, как выглядит эффективная стратегия тестирования API в реальных условиях.

6. Apache Spark - популярная система с открытым исходным кодом для обработки больших объемов данных. Большие данные могут быть большими — возможно, петабайтами. Он также может быть сложным, объединяющим несколько схем и источников. Это также может быть высокоскоростной поток данных. Все эти типы данных выигрывают от обширной параллельной обработки различных частей данных в памяти, чтобы свести к минимуму циклические перемещения в постоянное хранилище. Apache Spark облегчает это, автоматически масштабируя обработку и память для эффективного анализа больших наборов данных. Приложения для Apache Spark включают пакетную обработку фильтрации данных, агрегирование данных и преобразование данных в пригодные для использования наборы данных. Он также может использовать модели машинного обучения и анализа для обработки огромных объемов данных для создания моделей, поиска тенденций и прогнозирования будущих сценариев. Это также полезно для обработки данных в режиме реального времени, которая быстро обрабатывает и анализирует поток данных. Многие другие области применения Apache Spark ограничены только вашим воображением.

6 Контрольные вопросы:

12. Прочитайте рецензии на фильм «Поле битвы: Земля». Оцените вручную общую тональность каждой рецензии по пятибалльной шкале (1..5).Выделите аспекты фильма (сюжет, игра актеров, спецэффекты и т.п.) и оцените тональность рецензий по каждому из аспектов, подкрепите оценки примерами из текстов. Составьте вручную итоговый «реферат» оценок фильма на основе трех рецензий.
13. Лексическая семантика: отношения между словами.
14. WordNet: структура, состав, близость по графу.
15. Векторные представления семантики слов: PPMI на основе матрицы совместной встречаемости, GloVe, word2vec, fasttest.
16. Векторные представления семантики предложений: paragraph2vec, sent2vec, skip-thought vectors.
17. Подходы к оценке семантических представлений слов и предложений.
18. Анализ тональности: варианты постановки задачи, приложения и подходы к решению.
19. Методы автоматического расширения словарей тонально окрашенной лексики.
20. Анализ тональности на основе нейронных сетей.

Практическая работа №4 «Извлечение информации»

1 **Цель работы**: изучить способы извлечения информации из текстов.

2 **Порядок выполнения работы**:

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.

3 **Краткие теоретические сведения**

Значительная доля информации, доступной в электронном виде, представлена текстами на естественном языке. Заключенная в них полезная информация не структурирована, а значит, ее невозможно обработать и проанализировать классическими вычислительными методами и средствами.

Технология извлечения информации (ТИИ) из текстов на естественном языке позволяет автоматически просматривать относительно большой объем текстов, содержащих сравнительно небольшое количество искомой информации. Обнаруженная в тексте информация преобразуется в структурированный формат: выявляются целевые факты, объекты, отношения в виде, пригодном для дальнейшей автоматической обработки (статистической обработки, визуализации, поиска закономерностей в данных и др.).

Иногда ТИИ рассматривают как специфическую разновидность информационного поиска. Отличия ТИИ от информационного поиска заключаются в том, что запросы должны быть известны заранее, результатом же является не набор ссылок на документы, а построенные структуры данных, описывающие релевантные факты из набора документов.

Приведем некоторые области применения ТИИ:

– расширение возможностей информационного поиска (поиск не по ключевым словам, а по фактам, ситуациям, объектам, отношениям);
– построение досье на персон или организации из открытых текстовых источников;
– мониторинг сообщений СМИ (примеры событий, которые могут представлять интерес: слияние и поглощение компаний, появление новых игроков на рынке, выпуск новой продукции, теракты);
– извлечение специфической метаинформации из коллекций документов большого объема (например, построение реляционной БД с информацией о типах событий, объектах и субъектах по текстовой базе муниципальных нормативно-правовых актов, связанных с недвижимостью).

Первоначально задача ТИИ формулировалась как выделение фрагментов текста, содержащих релевантную информацию, и, возможно, преобразование их в реляционную форму. Для решения задачи в такой постановке часто достаточно анализировать локальный контекст, используя ограниченный набор

знаний предметной области. Назовем такую технологию извлечением информации в слабом смысле. Результаты извлечения информации в слабом смысле и характер их представления несколько ограничивают возможности дальнейшего использования добытых из текста данных. Извлечением информации в сильном смысле назовем переход от базы текстовых фактов к такому их представлению, которое можно было бы использовать как интеллектуальный информационный ресурс, своего рода базу текстовых знаний.

Исследования авторов были направлены на усовершенствование методов и расширение возможностей ТИИ, что позволило бы вплотную подойти к решению задачи извлечения информации в сильном смысле. Полигоном для экспериментальной проверки идей и практического воплощения разработанных подходов стала система ИСИДА-Т (интеллектуальная система извлечения данных и их анализа (для обработки текстов)), над которой ведется работа в течение нескольких лет.

Чтобы получить информацию из прочитанного фрагмента текста (понять текст), человек должен знать язык, на котором написан текст, и располагать некоторым объемом «фоновых» знаний. Аналогично система извлечения информации из текста должна располагать средствами анализа естественного языка и некоторым объемом знаний предметной области.

Краеугольным камнем системы ИСИДА-Т является точная настройка на предметную область и конкретную задачу извлечения. Это достигается за счет редактирования лингвистических ресурсов, ресурсов знаний, правил извлечения и правил трансформации. Настройка может потребовать также включение в процесс обработки дополнительных специализированных методов обработки текста. Кроме того, для каждой задачи необходимо подобрать наиболее подходящие алгоритмические средства анализа из набора имеющихся. Эти аспекты требуют создания такой архитектуры, при которой легко могут добавляться и замещаться алгоритмические компоненты процесса извлечения.

Конфигурирование на алгоритмическом уровне потребовало создания модульной архитектуры и декларативного подхода к определению процесса извлечения. Модули получили название обрабатывающих ресурсов в противовес лингвистическим ресурсам и ресурсам знаний. В конфигурации декларируются порядок обработки документа аналитическими модулями, потоки данных между ними, а также параметры их работы.

Обрабатывающие ресурсы можно разделить на следующие группы.

– Ресурсы предобработки. К ним относятся средства определения кодировки документа, извлечения текста и стилевой разметки из документа, предварительной фильтрации.

– Ресурсы лингвистического анализа. Осуществляют разбор текста на отдельные слова, морфологический анализ (в том числе специализированные варианты для различных категорий имен собственных), поверхностный синтаксический анализ и определение границ предложений.

– Ресурсы извлечения. Осуществляют поиск в документе целевой лексики и синтаксических конструкций, а также первичное структурирование информации.

– Ресурсы унификации знаний и вывода. Осуществляют унификацию и отождествление элементов знаний, вывод производных знаний.

– Ресурсы подготовки результата. Осуществляют приведение извлеченной информации к определенному формату и передачу за пределы последовательности обработки (в БД, глобальный ресурс знаний, файл, приложение).

Используемые в ТИИ средства для анализа естественного языка можно разделить на две большие категории: средства общего лингвистического анализа и предметно-ориентированные методы распознавания текстовых ситуаций.

Средства общего лингвистического анализа включают графематический, морфологический и синтаксический анализ. Эти средства применимы практически во всех предметных областях, существует ряд реализаций с довольно высокими показателями качества, поэтому останавливаться на них подробно нет необходимости.

Распознавание текстовых ситуаций состоит в выделении фрагментов текста, описывающих объекты, и содержательных связей между этими фрагментами, в той или иной мере основанных на синтаксисе естественного языка. Можно рассматривать распознавание ситуаций как ориентированный на предметную область частичный, но точный синтактико-семантический анализ.

Распознавание основывается на сопоставлении с образцом, заданным при помощи правил на специализированном формальном языке. Правила определяют не только образец, но и действия, которые должны быть выполнены при успешном сопоставлении. Правила работают не с текстом как последовательностью символов, а со структурами, построенными над текстом и выражающими лингвистическую и предметную информацию о нем.

Для упрощения конфигурирования системы желательно, чтобы все модули использовали одинаковый способ представления информации о тексте (разметки текста). Рассмотрим структуры данных, которые используются всеми модулями системы ИСИДА-Т, в том числе средствами общего лингвистического анализа.

В различных системах обработки текста на естественном языке используется широкий спектр средств для представления лингвистической и предметно-ориентированной информации о тексте в целом или его фрагментах. Единого подхода к представлению разметки текста и информации о нем не существует.

В последнее десятилетие довольно широко используется способ представления информации о тексте, основанный на так называемых аннотациях, отличающийся простотой и высокой степенью универсальности. Сегодня многие системы обработки текста в той или иной степени используют идеи модели аннотаций.

Аннотация – объект, который приписывается фрагменту текста (например, слову, словосочетанию, предложению, ссылке на сущность предметной области и т.д.) и описывает свойства этого фрагмента. Аннотации разбиты на конечное множество классов. Каждый класс аннотаций описывает текст в определенном аспекте. Информация о фрагменте представлена значениями именованных атрибутов аннотации. Наборы классов и атрибутов аннотаций намеренно не специфицированы, чтобы можно было использовать произвольный набор обрабатывающих модулей и представлять необходимую лингвистическую и предметную информацию. Обмен данными между модулями тоже идет в терминах аннотаций: новые аннотации могут строиться на основании полученных на предыдущих этапах анализа.

Представление информации с помощью аннотаций дает возможность разрабатывать средства анализа текста, компоненты которых слабо связаны между собой. Не отражающееся на функциональных характеристиках сложной системы уменьшение числа зависимостей между ее составляющими облегчает ее понимание, разработку и поддержку. Слабая связность – это существенное преимущество, так как она повышает возможность повторного использования компонентов и снижает риск критических сбоев, вызванных неправильным взаимодействием компонентов (например, из-за того, что в цепочке обработки какой-то компонент ошибочно не был зарегистрирован или же частично нарушился порядок обработки).

Впрочем, базовая модель аннотаций не лишена недостатков. В частности, в ней не предусмотрены средства проверки соответствия атрибутов и их значений. Атрибуты могут быть только атомарными. Отсутствует возможность установления связей между отдельными аннотациями. Нет средств для контроля расположения границ ан- нотаций разных классов, в то время как для большинства классов аннотаций можно задать условия, описывающие их взаимное расположение. Например, аннотации, описывающие синтаксис предложения в терминах системы составляющих, не могут пересекаться – для них возможно только отношение строгого вхождения или совпадения.

В реализации системы ИСИДА-Т модель аннотаций дополнена некоторыми полезными средствами. В частности, было снято ограничение на атомарность атрибутов и добавлена возможность устанавливать ссылки между аннотациями.

Для распознавания текстовых ситуаций используется набор правил, описывающих характерные для конкретной задачи способы выражения ситуации в тексте. Эти правила задают образец для сопоставления и действия, которые должны быть выполнены после успешного сопоставления. Качество работы (полнота и точность) ТИИ тесно связано с возможностями языка правил. Ряд современных систем извлечения информации (в том числе система ИСИДА-Т) берут за основу различные диалекты языка CPSL. Использование этого языка подразумевает разметку текста при помощи аннотаций.

Единицей трансляции языка правил является фаза. Правила, входящие в одну фазу, применяются в недетерминированном порядке. Результаты фазы –

изменения, внесенные в набор аннотаций после работы правил, – фиксируются после применения всех правил и становятся доступными в последующих фазах. Поэтому правило не может использовать результаты работы другого правила из этой же фазы. Можно рассматривать фазу как модуль для специфического анализа текста. Работа фаз может перемежаться применением произвольных обрабатывающих ресурсов.

Правило – основная единица языка. Правила представляются в виде «образец®действие». Здесь «образец» – образец для поиска в терминах высказываний о взаимном расположении и значениях атрибутов аннотаций разных классов (левая часть правила); «действие» – набор действий, выполняемых при успешном сопоставлении (правая часть правила). По структуре левая часть правила во многом схожа с регулярным выражением, но существенное отличие состоит в том, что роль символов в правиле играют тесты. Тест представляет собой конъюнкцию высказываний (элементарных тестов) о значениях атрибутов аннотаций разных классов. Из тестов могут образовываться сложные конструкции с использованием следования, альтернативы, квантификаторов и скобок. Чтобы обозначить границы фрагментов текста, сопоставленных подвыражениям, используются метки. Метка – это идентификатор, которым помечается образец. В дальнейшем (при выполнении действий в правой части правила) можно использовать метку для ссылки на фрагмент текста, сопоставленный подвыражению.

Язык правил, используемый в системе ИСИДА-Т, является расширением CPSL. Предлагаемые нами расширения преследуют две цели: обеспечить возможность описывать более сложные контексты, в которых встречается целевая информация, и снизить объем рутинной работы при создании системы правил за счет более компактного описания контекста. Расширения включают в себя поддержку дополнительных типов данных, большего числа квантификаторов и метасимволов. Помимо этого, реализована поддержка переменных и списков значений, существуют гибкие возможности проверки взаимного расположения аннотаций.

Общая проблема средств распознавания текстовых ситуаций – резкое снижение производительности. Для решения проблемы использовались два основных способа оптимизации интерпретатора правил: предобработка правил путем анализа потоков управления и сокращение перебора кандидатов при выполнении тестов. Внедрение этих модификаций позволило ускорить интерпретацию правил в среднем в 6 раз в зависимости от конфигурации системы и качества входных данных (в отдельных случаях наблюдался прирост производительности до двух порядков). В большинстве случаев повышение производительности сопровождалось снижением расхода памяти на 20–40 %.

Практически в любой предметной области для точного извлечения требуются априорные знания о ней – знания о понятиях, объектах и отношениях, связанных с целями извлечения или являющихся целями. В свою очередь, извлеченная из текстов информация может содержать новые знания о предметной области и быть полезной для дальнейшей автоматической

обработки текста. Тесная связь между априорной и извлеченной информацией, а также между предметными и лингвистическими знаниями сформировала потребность в унификации средств представления.

Представление знаний. Интегрированный ресурс знаний (РЗ) системы ИСИДА-Т объединяет в себе базу априорных предметных знаний, хранилище фактографической информации и словарь. Предметные знания хранятся в РЗ в структурах, называемых элементами знаний. Элементы знаний делятся на 4 категории: концепты (CT), экземпляры концептов (CI), типы предметных отношений (RT), экземпляры отношений (RI). В данной работе в подходе к представлению знаний используются элементы семантических сетей и систем фреймов.

Концепты и типы отношений служат для представления онтологической информации о предметной области и задаются априорно. Экземпляры концептов и отношений составляют базу фактов предметной области и могут быть как априорными, так и извлеченными из текстов.

Для каждого элемента знаний задается набор атрибутов. В списках атрибутов CT и RT хранятся пары «имя–ограничения на значение», в списках атрибутов CI и RI – пары «имя–значение». В терминах системы фреймов CT и RT выражались бы прототипами фреймов, а CI и RI – экзофреймами. Неявно определены два специальных (служебных) типа отношений: ISA и AKO. Их интерпретация такая же, как в системах фреймов.

Что именно представляет собой конвейер извлечения информации? Проще говоря, извлечение информации — это задача извлечения структурированной информации из неструктурированных данных, таких как текст.

Реализация конвейера извлечения информации состоит из четырех частей. На первом этапе пропускается вводимый текст через модель разрешения кореференции. Разрешение кореференции - это задача поиска всех выражений, которые относятся к определенному объекту. Проще говоря, он связывает все местоимения с упомянутым объектом. Как только этот шаг будет завершен, он разбивает текст на предложения и удаляет знаки препинания. Конкретная модель ML, используемая для связывания именованных сущностей, работает лучше, когда сначала удаляются знаки препинания. В части конвейера, связывающей именованные сущности, необходимо извлечь все упомянутые сущности и подключить их к целевой базе знаний. Целевой базой знаний в данном случае является Википедия. Связывание именованных сущностей полезно, поскольку оно также имеет дело с устранением неоднозначности сущностей, что может быть большой проблемой.

Как только извлечены упомянутые сущности, конвейер IE пытается определить отношения между сущностями, которые имеют смысл на основе контекста текста. Результаты конвейера IE представляют собой сущности и их взаимосвязи, поэтому имеет смысл использовать базу данных графиков для хранения выходных данных.

Используем следующий текст из Википедии, чтобы ознакомиться с конвейером IE.

«Илон Маск - бизнес-магнат, промышленный дизайнер и инженер. Он является основателем, генеральным директором, техническим директором и главным дизайнером SpaceX. Он также является ранним инвестором, генеральным директором и разработчиком продуктов Tesla, Inc. Он также является основателем компании Boeing и соучредителем Neuralink. Будучи многомиллиардером, Маск стал самым богатым человеком в мире в январе 2021 года, его состояние на тот момент оценивалось в 185 миллиардов долларов, превзойдя Джеффа Безоса. Маск родился в семье матери-канадки и отца-южноафриканца и вырос в Претории, Южная Африка. Он недолго учился в Университете Претории, а затем в возрасте 17 лет переехал в Канаду, чтобы поступить в Королевский университет. Два года спустя он перевелся в Пенсильванский университет, где получил двойную степень бакалавра по экономике и физике. Он переехал в Калифорнию в 1995 году, чтобы поступить в Стэнфордский университет, но вместо этого решил заняться деловой карьерой. Он стал соучредителем компании по разработке веб-программного обеспечения Zip2 вместе со своим братом Кимбалом Маском.»

Шаг 1. Разрешение кореференции

Как уже упоминалось, разрешение кореференции пытается найти все выражения в тексте, которые относятся к определенному объекту. В своей реализации можно использовать модель Neuralcoref от Huggingface, которая работает поверх платформы Spacy. Используя параметры модели Neuralcoref по умолчанию. Важно отметить, что модель Neuralcoref плохо работает с местоимениями местоположения. Код для части разрешения кореференции выглядит следующим образом:

```
import spacy
import neuralcoref


# Load SpaCy
nlp = spacy.load('en')
# Add neural coref to SpaCy's pipe
neuralcoref.add_to_pipe(nlp)


def coref_resolution(text):
    """Function that executes coreference resolution on a given text"""
    doc = nlp(text)
    # fetches tokens with whitespaces from spacy document
    tok_list = list(token.text_with_ws for token in doc)
```

```
    for cluster in doc._.coref_clusters:
        # get tokens from representative cluster name
        cluster_main_words = set(cluster.main.text.split(' '))
        for coref in cluster:
            if coref != cluster.main:  # if coreference element is not the representative element of that cluster
                if coref.text != cluster.main.text and bool(set(coref.text.split(' ')).intersection(cluster_main_words)) == False:
                    # if coreference element text and representative element text are not equal and none of the coreference element words are in representative element. This was done to handle nested coreference scenarios
                    tok_list[coref.start] = cluster.main.text + \
                        doc[coref.end-1].whitespace_
                    for i in range(coref.start+1, coref.end):
                        tok_list[i] = ""


    return "".join(tok_list)
```

Применив скрипт, написанный выше, к тексту должен полуяиться следующий результат:

Илон Маск также является ранним инвестором, генеральным директором и разработчиком продуктов Tesla, Inc. Илон Маск также является основателем компании Boeing и соучредителем Neuralink. Будучи многомиллиардером, Маск стал самым богатым человеком в мире в январе 2021 года, его состояние на тот момент оценивалось в 185 миллиардов долларов, превзойдя Джеффа Безоса. Маск родился в семье матери-канадки и отца-южноафриканца и вырос в Претории, Южная Африка. Илон Маск недолго учился в Университете Претории, а затем в возрасте 17 лет переехал в Канаду, чтобы поступить в Королевский университет. Два года спустя Илон Маск перевелся в Пенсильванский университет, где получил двойную степень бакалавра в области экономики и физики. Илон Маск переехал в Калифорнию в 1995 году, чтобы поступить в Стэнфордский университет, но вместо этого решил заняться деловой карьерой. Илон Маск стал соучредителем компании по разработке веб-программного обеспечения Zip2 вместе с братом Илона Маска Кимбалом Маском.

В этом примере не требуются продвинутые методы разрешения кореференции. Модель Neuralcoref изменила пару местоимений “Он” на “Илон Маск”. Хотя это может показаться очень простым, это важный шаг, который повысит общую эффективность нашего конвейера IE.

Шаг 2. Связывание именованных сущностей

Использование модели Facebook BLINK не является оптимальным, потому что она требует не менее 50 ГБ свободного места, что само по себе не является большой проблемой, но для этого также требуется 32 ГБ оперативной памяти. Поэтому самое оптимальное решение к использованию старого доброго API Wikifier, который уже показал свою полезность. И это совершенно бесплатно.

Прежде чем запустить вводимый текст через API Wikifier, необходимо разделить текст на предложения и удалить знаки препинания. В целом, код для этого шага выглядит следующим образом:

```
import
urllib
        from string import punctuation
        import nltk


        ENTITY_TYPES = ["human", "person", "company", "enterprise",
        "business", "geographic region",
                    "human settlement", "geographic entity", "territorial entity type",
        "organization"]


        def wikifier(text, lang="en", threshold=0.8):
           """Function that fetches entity linking results from wikifier.com API"""
           # Prepare the URL.
           data = urllib.parse.urlencode([
              ("text", text), ("lang", lang),
              ("userKey", "tgbdmkpmkluegqfbawcwjywieevmza"),
              ("pageRankSqThreshold", "%g" %
               threshold), ("applyPageRankSqThreshold", "true"),
              ("nTopDfValuesToIgnore", "100"), ("nWordsToIgnoreFromList",
        "100"),
              ("wikiDataClasses", "true"), ("wikiDataClassIds", "false"),
              ("support", "true"), ("ranges", "false"), ("minLinkFrequency", "2"),
              ("includeCosines", "false"), ("maxMentionEntropy", "3")
           ])
           url = "http://www.wikifier.org/annotate-article"
           # Call the Wikifier and read the response.
           req = urllib.request.Request(url, data=data.encode("utf8"),
        method="POST")
           with urllib.request.urlopen(req, timeout=60) as f:
              response = f.read()
```

```
        response = json.loads(response.decode("utf8"))
    # Output the annotations.
    results = list()
    for annotation in response["annotations"]:
        # Filter out desired entity classes
        if ('wikiDataClasses' in annotation) and (any([el['enLabel'] in
ENTITY_TYPES for el in annotation['wikiDataClasses']])):


            # Specify entity label
                if any([el['enLabel'] in ["human", "person"] for el in
annotation['wikiDataClasses']]):
                    label = 'Person'
                elif any([el['enLabel'] in ["company", "enterprise", "business",
"organization"] for el in annotation['wikiDataClasses']]):
                    label = 'Organization'
                elif any([el['enLabel'] in ["geographic region", "human
settlement", "geographic entity", "territorial entity type"] for el in
annotation['wikiDataClasses']]):
                    label = 'Location'
                else:
                    label = None


                results.append({'title': annotation['title'], 'wikiId':
annotation['wikiDataItemId'], 'label': label,
                                'characters': [(el['chFrom'], el['chTo']) for el in
annotation['support']]})
            return results
```

Приятной особенностью процесса викификации является то, что также можно получать соответствующие идентификаторы викиданных для объектов вместе с их названиями. Наличие идентификаторов викиданных решает проблему устранения неоднозначности сущности. Тогда можно задаться вопросом, что произойдет, если объект не существует в Википедии. В этом случае, к сожалению, Викификатор не распознает его. Однако в Википедии насчитывается более 100 миллионов сущностей.

Если внимательно посмотреть на результаты, то можно заметить, что Претория ошибочно классифицирована как Организация.

Шаг 3. Извлечение связей

Здесь будет рассмотрен процесс извлечения рабочих отношений. Хорошим решением на этом шаге будет использование проекта с открытым исходным кодом OpenNRE. В нем представлены пять моделей извлечения отношений, которые были обучены на базе данных Wiki80 или Tacred. Модели, обученные на наборе данных Wiki80, могут определять 80 типов отношений.

Если посмотреть на пример вызова извлечения отношений в библиотеке OpenNRE, можно заметить, что он только выводит отношения и не пытается извлечь именованные сущности. Необходимо предоставить пару сущностей с параметрами h и t, а затем модель попытается вывести взаимосвязь.

```
model.infer({'text': 'He was the son of Máel Dúin mac Máele Fithrich, and grandson of the high king Áed Uaridnach (died 612).', 'h': {'pos': (18, 46)}, 't': {'pos': (78, 91)}})
('father', 0.5108704566955566)
```

Результаты выводят тип взаимосвязи, а также уровень достоверности прогноза. Код для извлечения отношений выглядит так:

```
    # First get all the entities in the sentence
    entities = wikifier(sentence, threshold=entities_threshold)
    # Iterate over every permutation pair of entities
    for permutation in itertools.permutations(entities, 2):
        for source in permutation[0]['characters']:
            for target in permutation[1]['characters']:
                # Relationship extraction with OpenNRE
                data = relation_model.infer(
                    {'text': sentence, 'h': {'pos': [source[0], source[1] + 1]}, 't': {'pos': [target[0], target[1] + 1]}})
                if data[1] > relation_threshold:
                    relations_list.append(
                        {'source': permutation[0]['title'], 'target': permutation[1]['title'], 'type': data[0]})
```

Необходимо использовать результаты связывания именованной сущности в качестве входных данных для процесса извлечения отношений. Перебирается каждая перестановка пары сущностей и производится попытка вывести взаимосвязь. Как можно видеть по коду, также есть параметр relation_threshold, чтобы опустить отношения с небольшим уровнем достоверности.

Извлечение отношений - сложная задача для решения, поэтому идеальный результат получить сейчас невозможно. Этот конвейер IE работает так же хорошо, если не лучше, чем некоторые коммерческие решения. И очевидно, что другие коммерческие решения намного лучше.

Шаг 4. График знаний

Поскольку используются сущности и их взаимосвязи, имеет смысл хранить результаты только в базе данных графиков. Например, использовать Neo4j.

В графической визуализации легко заметить, что, хотя большинство взаимосвязей выводятся в обоих направлениях, это верно не во всех случаях. Например, связь между местом работы Илона Маска и Университетом Пенсильвании предполагается только в одном направлении. Это подводит к еще одному недостатку модели OpenNRE. Направление отношений не так точно, как бы хотелось.

Поскольку есть необходимость сохранять результаты в Neo4j, также придется загрузить и настроить его. Необходимо сохранить сущности и отношения на графике, но также сохранить исходный текст. Наличие контрольного журнала очень полезно в реальных сценариях, поскольку уже известно, что конвейер IE не идеален.

Возможно, было бы немного нелогично реорганизовывать отношения в промежуточный узел. Проблема, с которой можно столкнуться, заключается в том, что не может быть отношений, указывающих на другие отношения. Учитывая эту проблему, необходимо преобразовать отношения в промежуточный узел. Можно использовать свое воображение для создания лучших типов отношений и меток узлов, но это то, что есть.

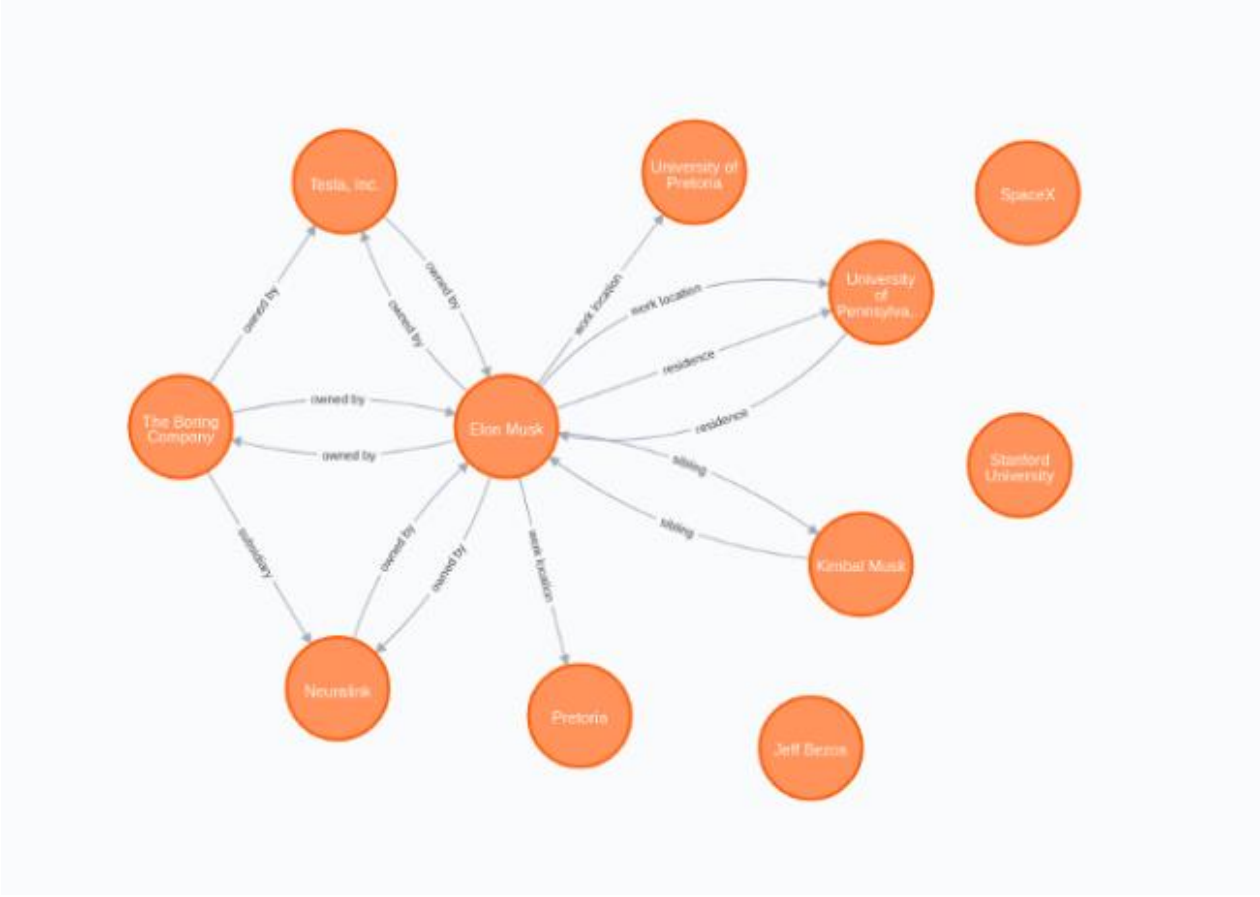

Рисунок 1 – Граф в Neo4j

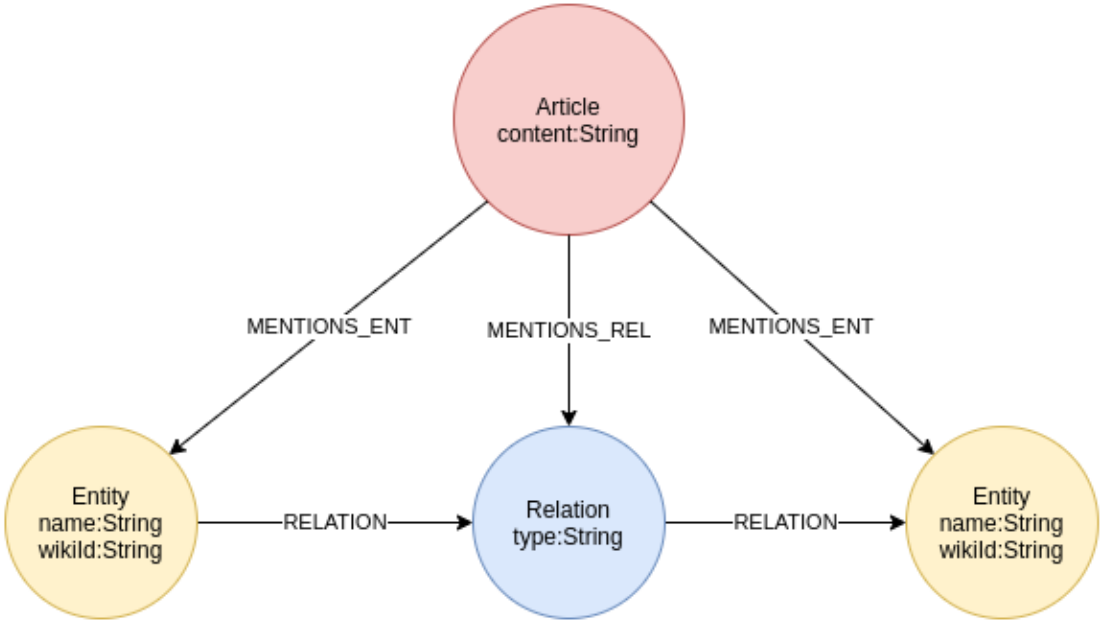

Рисунок 2 – Реорганизация схемы

Код для импорта 500 статей из набора данных новостей BBC в Neo4j следующий:

```
import urllib
import json
import pandas as pd
from neo4j import GraphDatabase


driver = GraphDatabase.driver('bolt://localhost:7687', auth=('neo4j', 'letmein'))


def ie_pipeline(text, relation_threshold=0.9, entities_threshold=0.8):
    # Prepare the URL.
    data = urllib.parse.urlencode([
        ("text", text), ("relation_threshold", relation_threshold),
        ("entities_threshold", entities_threshold)])

    url = "http://localhost:5000?" + data
    req = urllib.request.Request(url, data=data.encode("utf8"), method="GET")
    with urllib.request.urlopen(req, timeout=150) as f:
        response = f.read()
```

```
        response = json.loads(response.decode("utf8"))
    # Output the annotations.
    return response


import_refactored_query = """
UNWIND $params as value
CREATE (a:Article{content:value.content})
FOREACH (rel in value.ie.relations |
  MERGE (s:Entity{name:rel.source})
  MERGE (t:Entity{name:rel.target})
  MERGE (s)-[:RELATION]->(r:Relation{type:rel.type})-[:RELATION]->(t)
  MERGE (a)-[:MENTIONS_REL]->(r))
WITH value, a
UNWIND value.ie.entities as entity
MERGE (e:Entity{name:entity.title})
SET e.wikiId = entity.wikiId
MERGE (a)-[:MENTIONS_ENT]->(e)
WITH entity, e
CALL apoc.create.addLabels(e,[entity.label]) YIELD node
RETURN distinct 'done'
"""


with driver.session() as session:
    params = []
    for i,article in list(data.iterrows())[:500]:
        content = article['content']
        ie_data = ie_pipeline(content)
        params.append({'content':content, 'ie':ie_data})


        if (len(params) % 100 == 0):
            session.run(import_refactored_query, {'params':params})
            params = []


    session.run(update_query, {'params':params})
```

Извлечение семантических отношений из текстов предлагается разделить на три этапа:

1) извлечение из текстов сущностей и терминов;

2) извлечение поверхностных связей (триплетов) – пар сущностей, между которыми потенциально присутствует отношение, и фрагментов текстов, указывающих на наличие этого отношения;

3) группировка выделенных поверхностных связей с общей семантикой, из которых можно сформировать семантическое отношение. На первом этапе предлагается генерировать из синтаксических деревьев предложений большое количество вариантов сущностей – фрагментов текста, которые соответствуют предметам, процессам или явлениям из реального мира. При этом нас

интересуют не все факты, присутствующие в тексте, а только те связи и отношения, в которых участвуют термины предметной области – важные сущности для предметной области анализируемых текстов. Для оценки важности сущностей предлагается опираться на известные статистические подходы – различные модификации TF*IDF, и на небольшой набор эвристик, которые отфильтровывают заранее неперспективные варианты. Кроме того, при построении БЗ после извлечения сущностей необходима их нормализация, чтобы фрагментам с разной поверхностной лексико-синтаксической структурой сопоставить одну абстрактную сущность

Например, сущности «практикующий юрист» и «практический деятель в области права» необходимо сгруппировать. Для этого предлагается применить подходы агломеративной иерархической кластеризации именных групп. Для второго этапа извлечения поверхностных связей между выделенными сущностями предлагается адаптировать подходы систем типа ReVerb к русскому языку. Планируется модифицировать эвристики, заложенные в этих системах с учетом флективности и свободного порядка слов в русском языке. На третьем этапе группировку поверхностных связей предлагается осуществлять методами машинного обучения без учителя путем кластеризации признаковых описаний пар сущностей и их контекстов. Поверхностные связи, попавшие в один кластер, можно считать экземплярами некоторого неименованного отношения, выраженного в тексте. Если поверхностной связи не соответствует ни один из кластеров, то это означает, что отношение между парами сущностей устанавливать не следует. Во многих методах извлечения отношений из текстов на основе машинного обучения без учителя для подобной кластеризации используются вероятностные модели со скрытыми переменными.

Экспериментальные оценки качества работы таких моделей пока что значительно уступают оценкам методов на основе машинного обучения с учителем. Одним из возможных путей повышения качества моделей без учителя является расширение количества используемых признаков. Однако вместе с этим происходит усложнение структуры модели, что приводит к повышению трудоемкости ее обучения. Альтернативой усложнению структуры вероятностной модели является использование аддитивной регуляризации. Этот подход состоит в том, что помимо правдоподобия оптимизируются также дополнительные критерии-регуляризаторы, не обязательно вероятностной

природы. Как было показано ранее на примере регуляризации тематических моделей, такой подход позволяет строить многоцелевые модели, удовлетворяющие большому количеству дополнительных ограничений, либо композитные модели, объединяющие в себе несколько более простых моделей без значительного увеличения сложности алгоритмов обучения и вывода.

Типичная задача извлечения информации: просканировать набор документов, написанных на естественном языке, и наполнить базу данных выделенной полезной информацией. Современные подходы извлечения информации используют методы обработки естественного языка, направленные лишь на очень ограниченный набор тем (вопросов, проблем) — часто только на одну тему. Например, «Конференция по Пониманию сообщений» (en:Message Understanding Conference, MUC) — это конференция соревновательного характера и в прошлом она фокусировалась на таких вопросах:

- MUC-1 (1987), MUC-2 (1989): Военно-морские операции.
- MUC-3 (1991), MUC-4 (1992): Терроризм в латиноамериканских странах.
- MUC-5 (1993): Венчурные операции в области микроэлектроники.
- MUC-6 (1995): Новостные статьи об изменениях в управляющих процессах.
- MUC-7 (1998): Отчёты о запусках спутников.

Тексты на естественном языке могут потребовать некоего предварительного преобразования на язык (например, RDF — Resource Description Framework), понятный для компьютера.

Типичные подзадачи извлечения информации:

- Распознавание именованных элементов (сущностей), например: имён людей, названий организаций, географических названий, событий, временны́х и денежных обозначений и пр.
- Разрешение анафоры и кореференций: поиск связей, относящихся к одному и тому же объекту. Типичный случай таких ссылок — местоименная анафора.
- Выделение терминологии: нахождение для данного текста ключевых слов и словосочетаний (коллокаций).
- Автореферирование: выделение из текста смысловой, эмотивной, оценочной и пр. информации. Бывает генеративным и декларативным.

4 Задания для выполнения работы

– выделить в тексте именованные сущности двух типов - организация и персона.

– ответить на контрольные вопросы

Ввод: предложения, разделенные переносом строки.

Вывод: для каждого предложения из входных данных вывод в виде "индекс1 длина1 тег1 индекс2 длина2 тег2 ... индексN длинаN тегN EOL", где

индексK - индекс начала токена, длинаK - длина токена, тегK - тег токена. В качестве тегов могут выступать PERSON (персона) и ORG (организация). Если именованная сущность состоит из нескольких токенов, тег приписывается к каждому токену. Во входных предложениях могут отсутствовать именованные сущности этих типов. Разделитель в выводе в рамках одного предложения - пробельный символ, между выводами для различных предложений - перенос строки.

Sample Input:

```
Барак Обама принимает в Белом доме своего французского коллегу
Николя Саркози.
О возможном включении благотворительного фонда в список
"иностранных агентов" 7 мая написала газета «Ведомости».
```

Sample Output:

```
0 5 PERSON 6 5 PERSON 63 6 PERSON 70 7 PERSON EOL
22 18 ORG 41 5 ORG 93 6 ORG 101 9 ORG EOL
```

Методика выполнения

1. Для выполнения практического задания создать программу на языке Python(рисунок 1), которая выделяет в тексте именованные сущности.
2. Данный язык небходимо выбрать, так как его инструментарий содержит все необходимые инструменты для автоматического реферирования документов. В частности, использовать пакет natasha.

Алгоритм обработки текста показан на рисунке 3.

```python
def parseText(text):
    doc = Doc(text)
    doc.segment(segmenter)
    doc.tag_ner(ner_tagger)

    result = ''
    per = list(filter(lambda x: x.type == 'PER', doc.spans))
    org = list(filter(lambda x: x.type == 'ORG', doc.spans))
    spans = per + org

    for span in spans:
        for token in span.tokens:
            if hasattr(token, 'start'):
                result += str(token.start)
            else:
                result += str(0)

            result += ' ' + str(len(token.text))
            if span.type == 'PER':
                result += ' ' + 'PERSON '

            if span.type == 'ORG':
                result += ' ' + 'ORG '

    result += 'EOL'
    return result
```

Рисунок 3 – Алгоритм обработки текста

3. В результате работы програмы необходимо получать объект, в котором находятся обработанные данные(рисунок 4). Эти данные должны сохраняться в отдельный файл.

['13 8 PERSON 22 5 PERSON 30 2 PERSON 33 8 PERSON EOL', 'EOL', '77 4 PERSON 82 7 PERSON 103 6 PERSON 110 8 PERSON 161 8 PERSON 22 5 ORG 27 1 ORG 28 2 ORG 141 7 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', '17 10 PERSON EOL', 'EOL', '44 2 ORG 142 8 ORG EOL', '65 3 ORG EOL', '2 7 PERSON 10 7 PERSON EOL', 'EOL', '50 7 PERSON 58 6 PERSON 111 5 PERSON 117 12 PERSON 198 5 PERSON 204 7 PERSON EOL', '7 10 ORG EOL', '68 4 PERSON 73 10 PERSON EOL', 'EOL', 'EOL', '2 6 PERSON 9 7 PERSON EOL', 'EOL', '173 6 PERSON 180 9 PERSON 215 3 PERSON 219 9 PERSON 265 6 PERSON 272 8 PERSON 13 2 ORG 159 12 ORG 252 5 ORG 258 5 ORG EOL', '54 10 PERSON 93 3 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', '47 9 PERSON EOL', 'EOL', '101 9 ORG EOL', 'EOL', '106 6 PERSON 113 7 PERSON EOL', '57 4 ORG 64 10 ORG EOL', '0 7 PERSON 8 8 PERSON EOL', 'EOL', 'EOL', 'EOL', '69 10 ORG 80 7 ORG 88 2 ORG 91 13 ORG 105 5 ORG 111 1 ORG 112 4 ORG 116 1 ORG 137 10 ORG 148 9 ORG 158 4 ORG 179 3 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', '35 9 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', '53 6 PERSON 60 9 PERSON 43 8 ORG EOL', '0 4 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', '22 5 PERSON 286 PERSON 68 6 PERSON EOL', 'EOL', '0 9 PERSON EOL', '11 5 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', '94 4 PERSON 99 5 PERSON EOL', '10 7 PERSON EOL', '54 6 PERSON 61 5 PERSON 94 4 PERSON 99 8 PERSON 30 16 ORG 47 6 ORG EOL', '20 4 PERSON 25 9 PERSON EOL', '18 3 ORG EOL', 'EOL', 'EOL', 'EOL', '10 8 PERSON EOL', 'EOL', '88 7 PERSON 96 8 PERSON 12 6 ORG 19 5 ORG 27 4 ORG 32 2 ORG 37 5 ORG 67 7 ORG 75 11 ORG EOL', '47 2 ORG 67 12 ORG 80 12 ORG 93 2 ORG EOL', 'EOL', 'EOL', '84 2 ORG EOL', 'EOL', 'EOL', '139 PERSON EOL', '0 6 ORG 16 6 ORG EOL', '133 9 ORG EOL', '20 6 PERSON 49 9 ORG EOL', 'EOL', 'EOL', 'EOL', '15 3 ORG EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', '35 7 PERSON 43 5 PERSON 26 7 ORG 75 6 ORG EOL', '39 11 ORG 51 4 ORG EOL', 'EOL', 'EOL', 'EOL', '0 13 ORG 14 11 ORG EOL', '48 6 PERSON 55 6 PERSON 0 8 ORG 9 11 ORG 21 12 ORG 34 6 ORG EOL', '0 5 PERSON 6 4 PERSON EOL', 'EOL', 'EOL', '13 5 PERSON 19 9 PERSON EOL', 'EOL', '10 5 ORG 15 1 ORG 16 2 ORG EOL', 'EOL', '40 4 PERSON 45 9 PERSON EOL', '69 7 PERSON 77 8 PERSON 170 6 PERSON 177 7 PERSON 20 14 ORG 35 9 ORG 45 12 ORG 58 10 ORG EOL', '10 5 PERSON 16 11 PERSON EOL', '29 9 PERSON 39 6 PERSON 12 5 ORG EOL', 'EOL', '17 13 ORG EOL', '0 3 ORG EOL', 'EOL', '28 6 PERSON 35 7 PERSON EOL', '23 3 ORG EOL', '160 12 ORG 173 6 ORG EOL', '47 10 ORG EOL', 'EOL', '134 8 ORG 142 1 ORG 143 2 ORG EOL', 'EOL', '16 5 PERSON 2210 PERSON 6 3 ORG EOL', 'EOL', 'EOL', 'EOL', '19 7 PERSON 27 7 PERSON EOL', 'EOL', 'EOL', 'EOL', 'EOL', 'EOL', '10 7 PERSON 185 PERSON 55 6 PERSON 62 8 PERSON 44 6 ORG 95 10 ORG EOL', 'EOL', '24 7 PERSON 32 6 PERSON EOL', '54 5 PERSON 60 6 PERSON 47 4 ORG EOL', 'EOL', '13 6 PERSON EOL', '0 4 ORG EOL', '9 9 PERSON EOL', 'EOL', 'EOL', '119 7 PERSON 127 7 PERSON 153 5 PERSON 159 7 PERSON 115 3 ORG EOL', '0 10 ORG 11 10 ORG EOL', '66 7 PERSON EOL', '64 7 PERSON 72 7 PERSON EOL', '0 7 ORG 8 1 ORG 9 10 ORG 191 ORG EOL', 'EOL', '87 11 PERSON 27 8 ORG 36 6 ORG 47 9 ORG EOL', '13 8 ORG 22 5 ORG 69 13 ORG EOL', '42 8 PERSON 51 8 PERSON 19 6 ORG 26 5 ORG 32 9 ORG EOL', '0 5 PERSON 6 7 PERSON 51 6 ORG 58 5 ORG 186 6 ORG EOL', 'EOL', 'EOL', '1 1 ORG 2 1 ORG EOL',

Рисунок 4 – Результат работы программы

4. Проверить полученные данные(рисунок 5).

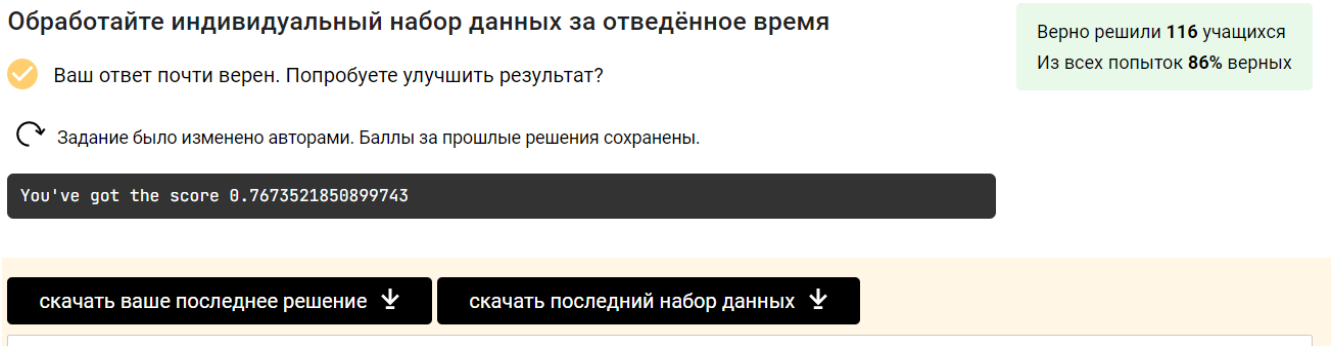

Рисунок 5 – Результат выполнения практического задания

5 Варианты заданий

1. Если у разработчика в его проекте есть модуль, который подключается к FTP-серверу для загрузки некоторых файлов, то разработчик напишет метод подключения к FTP через URL-адрес FTP и использует учетные данные, такие как имя пользователя и пароль для успешного подключения.

Если разработчик использует эти учетные данные в коде и жестко закодирует их в своем файле кода и развернет приложение, оно может работать нормально, работа выполнена! Теперь представьте, что через два месяца пароль для этого FTP-сайта изменен, и разработчик должен снова обновить этот пароль в вашем приложении, чтобы убедиться, что он не нарушает существующую функциональность FTP-соединения. Теперь в этом сценарии разработчик снова изменит код и повторно развернет его. Здесь разработчик должен для небольшого изменения конфигурации использовать последнюю версию кода, внести необходимые изменения, убедиться, что больше ничего не сломается, повторно развернуть приложение и затем протестировать его. И это может быть повторяющейся задачей через 2-3 месяца, когда какая-то конфигурация снова изменится. Что делать, если в приложении есть файл конфигурации, в котором есть пары ключ-значение “Пользователь”: “<имя пользователя>”, “Пароль”: “пароль”, и всякий раз, когда требуется какое-либо изменение, затрагивается только этот файл, и конфигурации обновляются, а не копаются в фактическом коде.

2. Исходя из опыта работы с .NET, мне было немного сложно понять, как разработчик может иметь файл конфигурации в приложении Python, который можно использовать для чтения значений настроек, и при этом не нужно даже прикасаться к коду для обновления или сохранения настроек. Конфигурационные файлы используются для хранения пар ключ-значение или некоторой настраиваемой информации, которая может быть прочитана или доступна в коде и в определенный момент времени. Если эта конфигурация изменится, разработчики могут просто изменить конфигурацию в этом файле конфигурации и не беспокоиться об изменении кода, так как это может потребовать повторной компиляции кода и его развертывания. Не только это, но и использование конфигурационных файлов делает ваши настройки и код более удобными для повторного использования, а также позволяет хранить информацию о настройках централизованно и отдельно. Конечно, конфиденциальная информация, такая как пароли, секреты и сертификаты, должна храниться в большей безопасности и может находиться в облачных хранилищах. Но основные настройки, используемые в приложении, могут быть частью файла конфигурации.

3. Графики знаний существуют уже почти полвека – этот термин был впервые введен в обиход в 1972 году! Долгое время они просто томились в академическом мире, пока Google не анонсировала свой график знаний в 2012 году. С тех пор графики знаний эволюционировали довольно резко, и теперь пути назад нет.За последние 10 лет наблюдается стремительный рост машинного обучения и искусственного интеллекта . Благодаря своей способности внедрять интеллектуальные данные в данные и добавлять контекст, графики знаний используются для того, чтобы сделать МО и ИИ более надежными, заслуживающими доверия и объяснимыми. Вполне естественно, что эти две технологии сближаются, стимулируя рост и необходимость графиков знаний. Графики знаний преобразуют

интеллектуальные данные в данные с помощью контекста и взаимосвязей. Графики знаний организуют и предоставляют единое место для поиска соответствующих данных и – что не менее важно – для интерпретации и принятия мер в соответствии с тем, что важно. Связанные данные, обогащенные смыслом, позволяют лучше отвечать на сложные запросы и находить наиболее эффективный путь к бесценной информации.

4. Инкапсуляция — это концепция объектно-ориентированного программирования, которая охватывает детали реализации класса. Почему бы не раскрыть детали реализации? Одна из хороших практик программирования заключается в том, что клиентский код должен иметь доступ только к открытым интерфейсам класса. Пока интерфейс остается прежним, клиентский код не нуждается в изменении в случае изменения реализации. Инкапсуляция также снижает сложность и упрощает процессы отладки. Кроме того, инкапсулированный класс помогает защитить данные и предотвратить неправильное использование, поскольку клиентский код не может получить доступ к защищенной части класса. В дополнение к деталям реализации, элементы данных, как правило, также являются участками кода, которые должны быть защищены. Однако может оказаться целесообразным предоставить общедоступный интерфейс, позволяющий клиентскому коду получать доступ к элементам данных в контексте класса. Этот тип интерфейса обычно называется функцией доступа. Функции доступа обычно бывают двух видов: геттер и сеттер. Получатель — это метод, который вызывается, когда мы обращаемся к элементу данных для чтения. В отличие от геттера, сеттер — это метод, вызываемый для изменения элемента данных.

5. "Тестирование кода" обычно организуется в виде пути, который начинается с модульного тестирования, продолжается интеграцией/сквозным тестированием и заканчивается приемочным тестированием пользователей. В начале этого пути модульное тестирование выполняет две вещи — оно подтверждает, что "тестируемый код" работет правильно при подаче своих входных данных. После этого вы переходите к интеграционному тестированию, где подтверждаете, что "передача" работает правильно (предположительно, только после модульного тестирования "следующая вещь в цепочке обработки", чтобы подтвердить, что она выполняет свою работу правильно). Поскольку API-интерфейсы являются интерфейсом к цепочке обработки, легко спутать модульное тестирование API с интеграционным тестированием всей этой цепочки обработки. Эффективная стратегия тестирования API предотвращает это. Чтобы продемонстрировать, как эффективно управлять тестированием API, я собираюсь рассмотреть стратегию модульного тестирования API (с использованием [Telerik Test Studio для API), чтобы продемонстрировать, как выглядит эффективная стратегия тестирования API в реальных условиях.

6. Apache Spark - популярная система с открытым исходным кодом для обработки больших объемов данных. Большие данные могут быть большими — возможно, петабайтами. Он также может быть сложным,

объединяющим несколько схем и источников. Это также может быть высокоскоростной поток данных. Все эти типы данных выигрывают от обширной параллельной обработки различных частей данных в памяти, чтобы свести к минимуму циклические перемещения в постоянное хранилище. Apache Spark облегчает это, автоматически масштабируя обработку и память для эффективного анализа больших наборов данных. Приложения для Apache Spark включают пакетную обработку фильтрации данных, агрегирование данных и преобразование данных в пригодные для использования наборы данных. Он также может использовать модели машинного обучения и анализа для обработки огромных объемов данных для создания моделей, поиска тенденций и прогнозирования будущих сценариев. Это также полезно для обработки данных в режиме реального времени, которая быстро обрабатывает и анализирует поток данных. Многие другие области применения Apache Spark ограничены только вашим воображением.

6 Контрольные вопросы:

1. Выделите вручную в новостном тексте следующие сущности: людей (PERSON), организации (ORG), географические объекты (GEO) и даты (DATE).
2. Схематизация документа и декодирование последовательности символов.
3. Какие виды подходов к распознаванию именованных сущностей вы знаете?

В чём особенность OpenLP для распознавания именованных сущностей?

Практическая работа №5 «Машинный перевод»

1 **Цель работы**: изучить способы машинного перевода текстов.

2 **Порядок выполнения работы**:

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.

3 **Краткие теоретические сведения**

Вплоть до конца 2016 года все продукты машинного перевода на рынке были основаны на алгоритмах, которые используют статистические методы для определения наилучшего возможного перевода для данного слова. Эта технология известна как Статистический машинный перевод (SMT). SMT включает в себя расширенный статистический анализ для оценки наилучших возможных переводов слова с учетом контекста нескольких окружающих слов.

С другой стороны, нейронный машинный перевод, или сокращенно NMT, выполняет этот процесс, пытаясь смоделировать абстракции высокого уровня в данных, гораздо ближе к тому, как это делается человеком, чем традиционный статистический подход. Нейронные сети лучше улавливают контекст полных предложений перед их переводом, что влечет за собой более высокое качество и более понятный для человека результат.

Прошли те времена, когда для перевода с одного языка на другой требовалось использование двуязычного словаря. В настоящее время, когда вы сталкиваетесь со словами на иностранном языке, вы заходите на платформу онлайн-перевода и мгновенно получаете результаты. По сути, это и есть машинный перевод - автоматизированный процесс перевода с одного языка на другой. Использование машинного перевода стало настолько распространенным явлением, что Google Translate сообщает, что он переводит более 100 миллиардов слов в день.

Помимо личного использования, машинный перевод (MT) помогает брендам и компаниям расширить свой охват глобальной аудитории. Сейчас больше, чем когда-либо, контент веб-сайта переводится на множество языков, чтобы помочь преодолеть языковой барьер. Поступая таким образом, они не только могут выйти на новые международные рынки, но также помогают демаршировать группы, которые изначально не были осведомлены о информации в Интернете.

Машинный перевод имеет преимущество перед человеческим переводом благодаря своей скорости и стоимости. С помощью компьютеров перевод происходит мгновенно, менее чем за треть стоимости. Несмотря на улучшение результатов MT, общее мнение профессионалов и бизнеса о том, что он не может заменить человеческий перевод, по-прежнему преобладает. Некоторые

компании предпочитают интегрировать свой процесс перевода, используя МТ для первоначального перевода, а затем выполняя некоторую редакторскую работу для дальнейшего повышения качества и точности. При правильном использовании машинный перевод может ускорить процесс перевода без ущерба для качества выходного контента.

Где необходимо перевести большие объемы контента. Перевод целых веб-сайтов или больших документов, особенно когда время является критическим фактором, исключительно в зависимости от человеческого перевода часто неосуществим. Машины могут работать без перерыва в течение длительного времени, обеспечивая результаты практически мгновенно. Переводчики-люди могут затем редактировать и просматривать машинные переводы, чтобы убедиться, что конечный контент отточен и точен.

Где нюансы не имеют значения. Язык по своей природе часто имеет неоднозначные правила; поэтому ограничение слов определенными стилями может быть затруднено. Однако язык, используемый в руководствах или документации по программному обеспечению, часто прост, что делает использование МТ целесообразным. Машинные переводчики работают, переводя отдельные слова или предложения параллельно. В результате получается набор последовательно переведенных предложений, а не связно переведенный текст. Проблемы с потоком, беглостью и удобочитаемостью часто присущи текстам с машинным переводом. Это особенно верно для технической или творческой работы, где использование слов в основном зависит от контекста.

Там, где бюджет перевода ограничен. Компании всегда находятся в поиске методов, которые позволят сократить их расходы. Вот почему многие небольшие фирмы обращаются к машинному переводу для своей работы - машины стоят дешевле, чем человеческий труд. Чтобы снизить затраты и сохранить целостность переведенного текста, некоторые компании предпочитают использовать сочетание как машинных, так и человеческих переводчиков.

Где содержание эфемерно. Контент, который может быть изменен, такой как отзывы клиентов, электронные письма или часто задаваемые вопросы, создается быстро и используется только один раз. Их качество не обязательно должно быть таким высоким, как у профессиональных документов. Качество перевода также не имеет значения, когда он используется для внутренних исследований.

Машинные переводы значительно улучшились за эти годы и помогают многим фирмам локализовать свой контент для охвата глобальной аудитории. При правильном использовании они могут производить продукцию высокого уровня с минимальным участием человека. Хотя они далеки от того, чтобы использоваться независимо, MTS полезны для больших объемов контента, где перевод человеком может быть невозможен.

Машинный перевод крайне важен в наше время, в данной статье подробно раскрываются основные достижения за всё время существования

машинного перевода. Рассмотрены нынешние системы машинного перевода, которые используются на практике. Растущий интерес к статистическому подходу машинного перевода зависит от разработки эффективных алгоритмов для обучения ранее предложенных вероятностных моделей. Однако одной из открытых проблем статистического машинного перевода является разработка эффективных алгоритмов для перевода заданной входной строки. Датой возникновения машинного перевода как научного направления выступает 1946 год. Впервые такая концепция была сформулирована директором отделения естественных наук Рокфеллерского фонда Уорреном Уивером, которая впоследствии вызвала большое количество дискуссий среди исследователей по всему миру. Первая конференция, на которой были представлены положения о машинном переводе, состоялась в 1952 году. Исследователи из многих стран мира собрались в Массачусетском университете. В течение двух лет после проведения конференции знания систематизировались, и лишь в 1954 году вниманию мирового сообщества была представлена первая концепция машинного перевода в Нью-Йорке.

При создании первой концепции машине подавалось около 60 предложений для перевода на русском языке, которые требовалось в короткий срок перевести на английский. Русский язык был выбран не случайно – ситуация на мировой арене предвещала начало холодной войны, поэтому требовалось ежедневно проводить анализ больших объемов информации и осуществлять переводы. В основном переводимые предложения имели узконаправленную тематику. Уже в начале 60-х годов 20 года исследователи из Соединенных Штатов Америки активно работали над модернизацией системы машинного перевода.

Исследования продолжились в начале 70-х годов, когда появлялись первые микрокомпьютеры и было начато развитие компьютерных сетей. В результате проведенной модернизации мировому сообществу было представлено устройство для машинного перевода второго поколения. Работа машины была выстроена на основе синтаксической структуры, росло количество правил орфографии, применяемых при переводе. В процессе перевода текста машина проводила преобразование текста согласно синтактической структуре языка, на который производился перевод, после чего слова подставлялись из словаря, который был значительно расширен в модели нового поколения. Следующий этап развития машинного перевода относится к 80-90 годам. Устройство третьего поколения осуществляло работу, основываясь не только на орфографических и морфологических правилах, но и на синтаксическом анализе, в результате чего качество перевода текста значительно возросло.

При переводе зачастую возникали проблемы с семантикой текста. В дальнейшем машина была усовершенствована – на рынке появились устройства семантического типа, которые отличались наилучшим качеством переведенного текста. На сегодняшний день все свершаемые переводы могут быть классифицированы по:

1.Жанрово-стилистическими особенностям: выделяются три функциональных разновидности перевода специальный, художественный и общественно-политический.

2.По форме перевода. В настоящее время наибольшую распространенность и популярность получили машинные переводы, основанные на правилах и статистике.

Системы перевода, действующие на основе правил, подразделяются на две категории: интерлингвистические и трансферные. Трансферные системы выполняют перевод в три основных этапа: производится анализ первоначального текста, дальнейший трансфер и конченый синтез. В качестве примера трансфертной системы перевода может быть названа система PROMT. Алгоритм машинного перевода, основанного на лингвистическом анализе, сводится к восьми основным этапам.

Данный алгоритм базируется на идее существования метаязыка. На перовом этапе машина получает исходное предложение или текст, представленный в виде файла. Следующий шаг – система разбивает полученный текст на предложения и слова. Среди наиболее сложных элементов для восприятия машиной выступают прямая речь, общепринятые и малораспространённые сокращения, имена, инициалы. Машина распознает слова, используя специальные шаблоны, которые представлены в виде буквенных и цифровых групп, а также знаков препинания и пунктуации. В процессе перевода как отдельные слова выделяются даты, сокращения. Например, слово «багрово-красный» будет производиться с учетом правил морфологического преобразования прилагательных.

Данный этап также включает в себя процедуру нормализации слов для того, чтобы осуществить их подготовку для поиска по внутреннему словарю. На третьем этапе производится морфологический анализ, который осуществляется с учетом особенностей языка, с которого переводится текст. Каждое слово в тексте ищется в словаре, после чего ему присваиваются лексическо-грамматические характеристики, например – число, род, падеж. Следующий шаг – проведение синтаксического анализа. Для каждого отдельного слова система проводит поиск слова, с которым оно должно быть согласовано после проведения перевода.

Основное снятие многозначности осуществляется при выполнении поиска главных слов. В дальнейшем синтаксическое дерево выстраивается при выполнении процесса распознавания лингвистических шаблонов, которые были заданы до выполнения перевода. Процедура распознавания шаблонов проводится в несколько этапов:

1.Осуществляется проверка того или иного слова на определенную часть речи.

2.Система выясняет, не является ли проверяемое слово омонимом.

3.Производится проверка на соответствие окончаний двух зависимых слов.

4.Система получает семантические характеристики для управления предлогов и глаголов.

В случае, если система распознает какой-либо шаблон, существует несколько вариантов дальнейшей работы над элементами, которые шаблон покрывает:

1.Исключение из списка лексико-грамматических классов слов, которые не удовлетворяют заданным условиям.

2.Исключение слова из выделенного множества отдельных слов в выбранном предложении, а также дальнейшее присоединение к нему главного слова. Таким образом, два слова согласуются и становятся зависимыми друг от друга.

3.Осуществляется фиксации между словами.

На пятом этапе машина анализирует полученный текст с точки зрения семантики для разрешения многозначности слов на основе уже полученного дерева зависимостей. В начале проведения семантического анализа производится разрешение многозначности слов, являющихся базовыми. После проведения процедуры машина проведет согласование зависимых слов. На шестом этапе машина осуществляет перевод уже построенного дерева, который включает в себя следующий перечень действий:

1.Базовые слова дерева переводятся пословно.

2.Глаголы, включенные в перечень базовых слов, обладающие изначальными характеристиками в виде рода, переводятся в совокупность глаголов одной парадигмы. Остальные глаголы не изменяются.

3.В результате выполнения перевода зависимые слова отражаются в виде совокупности различных вариантов.

Лишь на этапе проведения синтеза происходит определение требуемых лексических характеристик. На последнем этапе все части переведенного дерева зависимостей согласовываются между собой – для каждого из слов текста в словаре машина производит поиск нужной словоформы согласно классу. Среди явных преимуществ использования таких систем – повышенная точность переведенного текста с минимальным содержанием неестественности. Статистические системы получили широкую распространенность лишь в конце 20 века. При таком методе система обучается при помощи предоставления большого количества текстов на различных языках. При этом исходная информация одинакова по содержанию и структуре.

Так, методология системы «Яндекс.Переводчик» основывается на выполнении трех последовательных этапов:

1.Модели переводов, представляющей собой своеобразную таблицу, которая содержит информацию обо всех словах языка, на который требуется совершить перевод. При переводе текста система осуществляет учет не только отдельных слов, но и различных оборотов.

2.Модели языка, представляющей собой список наиболее встречаемых слов и оборотов, которые используются в том или ином тексте с определенной частотой.

3.Декодера, при котором выполняется поиск различных вариантов перевода того или иного текста.

При этом исходная модель языка как бы «дает подсказку» декодеру, какой из вариантов перевода соответствует той или иной фразе больше всего. Среди ключевых достоинств статистических систем для перевода – качество преобразованного текста, а также постоянное расширение перечня словарного запаса. Если в языке появляется какая-либо новая словоформа, система самостоятельно обучается и впоследствии может распознавать их при переводе. На сегодняшний день развитие машинных систем перевода не стоит на месте: проводятся разработки с использованием современных информационных технологий, применяется корпусная лингвистика.

Системы машинного перевода делятся на три категории: системы на основе грамматических правил (Rule-Based Machine Translation, RBMT), статистические системы (Statistical Machine Translation, SMT) и наиболее перспективные гибридные системы, сочетающие преимущества тех и других. Системы RBMT анализируют текст и строят его перевод на базе встроенных словарей и набора правил для данной языковой пары. В системах SMT применяется принцип статистического анализа: в программу загружаются большие (в миллионы слов) объёмы текстов на исходном языке и их переводы, выполненные человеком. Программа анализирует статистику межъязыковых соответствий, словоупотребления, синтаксических конструкций и т. д., и позже опирается на неё при выборе вариантов перевода — этот процесс называется самообучением. Систему может обучать и человек, корректируя выдаваемые переводы. Именно так работает широко известный сервис «Переводчик Google». Благодаря способности статистических и гибридных систем МП обучаться, накапливая языковые данные, качество перевода у них повышается с каждым переведённым текстом.

Главное преимущество машинного перевода в том, что он позволяет быстро справиться с очень большими объёмами текста и поэтому иногда оказывается экономически выгоднее перевода вручную. При этом следует помнить, что качество машинного перевода всегда будет уступать человеческому. Поэтому использовать его целесообразно лишь в определённых случаях

Во-первых, машинным способом могут переводиться материалы для внутреннего пользования, например, когда требуется в общих чертах понять содержание сайта, статей или писем на иностранном языке или найти сообщения на ту или иную тему в прессе на нескольких языках мира. Во-вторых, технические и узкоспециальные тексты, которые затем пойдут на редактирование специалистам по данной тематике — в этом случае машинный перевод используется в качестве подстрочника, на основе которого технический специалист будет создавать финальный текст, опираясь на свои знания в предметной области.

Многие типы материалов в принципе не предназначены для машинного перевода. Так, нельзя доверять машине тексты, где неточность перевода может

поставить под угрозу здоровье человека, работоспособность сложного прибора или крупный контракт — сэкономленное время здесь не оправдывает риска. Любые документы, подразумевающие юридическую ответственность (договоры, гарантийные обязательства), требуют контроля человека. Машинный перевод непригоден для маркетинговых материалов, где текст фактически переосмысливается в новом культурном контексте и создается заново.

В целом, приемлемого качества можно ожидать при переводе строго формализованных технических текстов, в то время как художественные и рекламные тексты машинному переводу не поддаются.

Предварительная подготовка материалов может значительно упростить задачу системе МП и редакторам, которые будут дорабатывать сырой машинный перевод. Такая подготовка начинается ещё на этапе написания исходного текста — с этой целью для технических писателей и авторов разрабатываются стандарты, соблюдение которых позволяет сделать текст более простым для понимания и перевода, как машинного, так и человеческого.

Существуют три правила, выполнение которых наиболее существенно повышает качество машинного перевода с английского языка.

1. Использование глаголов в действительном залоге вместо герундия
2. Использование активного залога вместо пассивного
3. Отказ от использования составных предложений и однородных членов

В идеале каждое предложение должно содержать одну законченную мысль. Именно это правило, в равной степени применимое для всех языков, является самым эффективным из трёх.

Как показал эксперимент, следование этим несложным правилам в сочетании с надлежащей адаптацией системы МП значительно повышает скорость доработки полученного на выходе текста. Это позволяет судить о том, сколь большую выгоду способна принести формализация и стандартизация текста при подготовке к машинному переводу — будь то управление составлением текста с помощью специальных программ, его предварительное редактирование или простое соблюдение автором нескольких наиболее эффективных правил.

Постредактирование — это правка сырого машинного перевода редактором, обычно имеющим специальную подготовку и опыт работы с машинными текстами. В большинстве случаев машинный перевод нуждается в последующей редакторской доработке, но иногда её можно опустить — в частности, когда тексты переводятся для внутренних нужд с целью понять общее содержание или найти определённые материалы. Затраты времени и труда на постредактирование — один из важнейших факторов, который следует учитывать, оценивая экономическую целесообразность машинного перевода. Художественные, рекламные и другие тексты, изначально не предназначенные для перевода машинным способом, не подлежат и постредактированию: чтобы довести качество текста до уровня, аналогичного человеческому переводу,

редактору придётся переписывать его практически с нуля, что сводит на нет всякую выгоду от применения машинного перевода.

Прибегая к машинному переводу, важно не только чётко представлять себе желаемый результат и понимать ограничения этого метода, но и учитывать ещё один фактор. Системы МП обычно требуют сложной индивидуальной настройки и доработки, в том числе «обучения» по конкретной тематике — без этого они показывают гораздо худшие результаты. В связи с этим машинный перевод имеет смысл использовать, только если предстоит перевести огромные объёмы однотипных текстов. В этом случае будет экономически целесообразно затратить определённое время на обучение системы, затем применить машинный перевод и получить на выходе текст, пригодный для постредактирования. Если же речь идёт о нескольких десятках страниц, пытаться внедрить машинный перевод бессмысленно и попросту убыточно.

Таким образом, машинный перевод с постредактированием может оказаться действительно выгодным, если переводятся тексты подходящего типа в очень больших объёмах. Поскольку большие объёмы переводов проходят через переводческие компании, которые часто специализируются в конкретных предметных областях, внедрение достаточно эффективных, но дорогих систем машинного перевода последнего поколения экономически оправдано именно в таких компаниях: ни поставщики контента, пусть даже крупные, ни индивидуальные переводчики не смогут самостоятельно эффективно использовать машинный перевод.

В 1954 году американцы продемонстрировали миру первую действующую программу машинного перевода. Ее разработали совместно фирма IBM и Джорджтаунский университет. В честь университета презентацию программы назвали Джорджтаунским экспериментом. На глазах у зрителей компьютер перевел 49 заранее отобранных предложений с русского на английский язык. Он использовала словарь из 250 слов и грамматику, состоящую из шести синтаксических правил. Долгое время все системы машинного перевода разрабатывали именно этот подход — в машину загружались все более сложные словари и правила анализа слов и предложений. Принято выделять три разных типа перевода на основе правил:

– трансферные системы (исходное предложение → препарированное исходное предложение → предложение на другом языке);

– системы пословного перевода (слова исходного текста → слова перевода);

– интерлингвистичесие системы (исходный текст → описание его смысла на универсальном языке-посреднике → текст перевода).

Трансферная система (transfer-based machine translation, transfer systems) в наибольшей степени подражает человеку-переводчику. Она анализирует исходное предложение, переводит его слова, выясняет их роли, а затем с помощью грамматики собирает новое предложение на конечном языке.

Система пословного перевода (word-by-word machine translation, другие названия — direct systems, dictionary based machine translation) — это самый

примитивный подход. В предложении переводятся все слова, а текст правится минимально. При этом получается перевод хотя и не очень гладкий, но в большинстве случаев понятный. Этот подход использовался в ранних программах, он и сейчас иногда применяется, например, для упрощения работы переводчиков-людей.

Интерлингвистичесие системы (interlingual systems) используют язык-посредник, который называется интерлингвой (interlingua). Строго говоря, это даже не язык в привычном для нас смысле, а некоторое формальное представление смысла человеческой речи и мыслей. На заре компьютерной лингвистики идея создания такой системы была очень популярна [Беляева, Откупщикова 1996], но создать полноценный универсальный вспомогательный язык ученым пока не удалось. Пословные переводчики и интерлингвистические системы сейчас практически не используются. Большинство RBMT систем применяют трансферный подход, поэтому рассмотрим его чуть подробнее.

Практически все компьютерные системы переводят текст по предложениям, поэтому текст должен быть разбит на предложения. В каждом из них надо выделить входящие в него слова, знаки препинания и другие элементы. Полученные элементы называются токенами (tokens), а сам процесс — токенизацией (tokenization).

Далее токены (то есть, слова и знаки) необходимо проанализировать, чтобы понять их роль в предложении. В результате для каждого слова будут определена его часть речи и его грамматические значения. Кроме того, для целей перевода необходимо сопоставить каждое слово с нужным разделом компьютерного словаря.

После того, как все токены проанализированы, необходимо понять структуру предложения — найти связи между словами, объединить слова в группы. Так, многие переводчики на раннем этапе собирают именные группы. Например, сочетание Добрые животные будет отмечено как целостная именная группа.

Когда структура предложения будет ясна, компьютер строит новое предложение на конечном языке, используя переводы слов. Но это в идеале. На практике всегда оказывается, что существующих правил обработки недостаточно, а описание слов в словаре должно быть более подробным. После этого начинается этап совершенствования системы. Разработчик анализирует переводы и пополняет словари и правила новыми условиями, чтобы компьютер мог учитывать смысловые и формальные связи. Например, английское слово plant имеет два значения: растение и завод, поэтому сочетание the strike at the plant может быть ошибочно переведено как забастовка на растении. Но предлог at в сочетании at the plant употребляется обычно тогда, когда речь идет о заводе. Поэтому можно добавить в систему правило: если слово plant сочетается с предлогом at, значит, переводить его надо как завод.

Правильный вариант перевода можно выбирать на основе семантического типа слов. Например, если английский глагол to press употребляется с

элементом одежды (the dress — «платье»), то его надо переводить как «гладить» или «погладить», а не «нажимать».

Улучшить качество текста помогает база памяти переводов (translation memory) — сохраненные заранее сегменты текста и их переводы, выполненные человеком. Это могут сочетания слов и целые предложения, например, идиомы или фрагменты деловой и личной переписки. При переводе они могут вставляться в текст автоматически или по указанию пользователя.

Способы лингвистической настройки перевода весьма разнообразны, но они требуют много часов (точнее, много лет) работы квалифицированных лингвистов. Наверное, самые подробные словари для русского языка существуют в компании PROMT. Они разрабатываются уже более 20 лет усилиями многих лингвистов и инженеров и могут быть интегрированы не только в системы перевода, но и в другие системы обработки языка.

Рассмотрим систему словарей компании PROMT — ведущей российской организации, занимающейся машинным переводом. Сейчас переводчики PROMT используют гибридные технологии, но их основу составляет мощная система словарей и грамматик, позволяющая компании выигрывать международные конкурсы по машинному переводу на русский язык. Система словарей в PROMTe содержит:

1. словари основ;
2. таблицы флексий;
3. вспомогательные таблицы.

В словаре основ хранится базовая форма каждого слова, его псевдооснова его грамматические и семантические признаки, а также его переводы на другой язык с подсказками, какой перевод в каких случаях использовать.

Как, например, устроена словарная статья для слова площадь? Она хранит его базовую форму площадь, чтобы было понятно, о каком слове идет речь. Кроме того, в словарной статье хранится псевдооснова площад — именно эта часть слова не меняется при изменении числа и падежа. Некоторые грамматические признаки этого слова записываются в словарную статью, а другие признаки хранятся в виде ссылок. В словарную статью можно записать, что это существительное мужского рода. И можно дать ссылку на ту строчку в таблице флексий, где хранятся окончания этого слова для разных падежей. Это будет та же строчка, что и у слова кровать, потому два этих слова изменяются одинаково.

Семантические признаки удобно привязывать к переводу. Английский слово площадь можно перевести тремя способами: area (если речь идет о территории), square (если подразумевается городская площадь) или ploschad (при транслитерации названий). Контекст слова площадь может подсказывать, как правильно его перевести. Например, если за ним идут цифры, а затем единицы измерения (100 кв.м.), то тогда это сочетание переводится как square of. Вся эта информация хранится в словаре основ.

В таблице флексий хранятся все типы изменения слов.

Вспомогательные таблицы созданы для обработки нестандартных случаев слов. Таких таблиц несколько. Это базы префиксов и постфиксов, то есть, начальных и конечных частей слов, а также база имен и географических названий.

В базе префиксов указаны, например, переводы начальных частей слов:

восточно — East
восьмидесяти — eighty
высоко — high
вэб — web

В базе постфиксов хранятся не только окончания слов, но и их предположительные части речи, а также похожее слово с таким же окончанием. Вот примеры записей из таблицы постфиксов:

*ойл — NOUN — Казахойл
*инвест — NOUN — Связьинвест
*дума — NOUN — Мосгордума
*банк — NOUN — Райффайзенбанк

В базе имен и географических названий указаны переводы многих личных имен с указанием их рода, типа склонения и некоторых других полезных сведений. В процессе лингвистической настройки словари активно пополняются, в них заносятся новые значения и условия. Но, к сожалению, эти кропотливо собранные данные можно использовать только для одной языковой пары (например, для перевода с английского на русский) и, как правило, только для одной системы. В этом отношении данные для статистического перевода гораздо более универсальны.

В основе статистического перевода лежат разработки американского математика Клода Шеннона, которые он вел в совершенно другой области. Во время Второй мировой войны немцы научились расшифровывать телефонные переговоры между правительствами США и Великобритании. Тогда решено было разработать новую систему передачи голоса по телефону. Предполагалось сжимать речевой сигнал и маскировать его с помощью шума. Клод Шеннон придумал математический аппарат, который позволял дешифровать полученный на выходе звуковой поток и выделять из шума исходный сигнал.

Разработки Шеннона позже пригодились и для машинного перевода. Ведь можно представить, что имеющееся исходной предложение получили в результате маскировки сигнала с помощью шума. А его перевод — это на самом деле исходное переданное нам предложение, которое нужно декодировать.

Если у нас есть предложение x в исходном языке, то задача машинного сводится к поиску в конечном языке такого предложения y, которое с наибольшей вероятностью является переводом предложения x. Иными словами, нас интересует условная вероятность — вероятность предложения y при наличии предложения x. Она записывается как p(y|x).

Например, перевести английское предложение The dog is hungry. Теоретически, его переводом может оказаться любое русское предложение y:

y1 = Собака сидит на крыше.

y2 =Собака голодная.

y3 =Собака голодный.

y4 = Собака есть голодная.

y5 = Голодная голодная собака.

…

У каждого из этих предложений есть своя вероятность быть переводом предложения x. Можно записать эти вероятности как p(y1|x), p(y2|x), p(y3|x) и т.д.

Из всех этих предложений yn компьютер должен выбрать самое вероятное, при условии, что у нас есть предложение x. Или, если записать это с помощью формул, ищем вот что:

(1) arg maxy p(y|x)

Но эту формулу можно переписать с помощью теоремы Байеса, о которой говорилось в пятой главе, посвященной методам машинного обучения.

(2) arg max y p(y|x) = arg max y p(y)×p(*x*|y) p(x)

Получившуюся формулу можно упростить. Значение в знаменателе p(x) — это вероятность появления исходного предложения x. Она является постоянной величиной и не влияет на поиск такого y, при котором вероятностная формула принимает максимальное значение. Поэтому знаменатель p(x) можно убрать:

(3) arg max y p(y|x) = arg max y p(y) × p(x|y)

Иными словами, нам необходимо найти, при каком y получается максимальное произведение двух величин. Первая из них — это вероятность p(y), вероятность появления предложения y в языке. Понятно, что такая вероятность будет выше у предложения Собака голодная, чем у предложения Собака голодный. Эта вероятность вычисляется с помощью модели языка, которой будет посвящен следующий раздел.

Множитель p(x|y) в формуле (3) — это «обратный перевод», вероятность того, что конечное предложение y можно перевести с помощью исходного предложения x. В нашем примере это вероятность того, что предложение Собака голодный можно перевести на английский как The dog is hungry.

Но тут может возникнуть вопрос. Неужели формула Байеса упрощает ситуацию? Ведь фактически, нам надо считать то же самое, вероятность перевода с одного языка на другой, но только в другую сторону. Но здесь

нужно вспомнить, что теоретически в качестве кандидата у нас может появиться абсолютно любое предложение, составленное из слов конечного языка. Кто знает, что нам предложит машина! Параметр p(y) позволяет нам оценить, насколько естественно звучит получившееся предложение.

Модель языка определяет вероятности появления того или иного предложения. Ведь в русском языке можно сказать не только Собака виляет хвостом, но и Хвост виляет собакой, а также Хвосты виляет хвосты или, допустим, Хвост хвост хвост. Никто из нас не может гарантировать, что ему никогда в жизни не встретится предложение Хвост хвост хвост.

Но вероятность появления этих предложений в языке разная. Очевидно, что Собака виляет хвостом — это более вероятное фраза, чем Собака виляет шлейфом.

Компьютер тоже может запомнить, насколько вероятно появление того или иного предложения в языке. Для этого ему надо «набраться языкового опыта» — посмотреть, какие предложения и словосочетания встречаются в языке. Чем больше текстов он «посмотрит», тем лучше сможет предсказать вероятность появления какого-либо предложения. Причем, в идеале эта вероятность никогда не должна равняться нулю. Разве можно быть уверенным, что какое-то предложение никогда и нигде не встретится?

Модель языка или языковая модель (language model) — это способ вычислять вероятность для всех теоретически возможных предложений языка. Эти способы могут быть разными, поэтому и модели языка существуют разные. Чтобы построить модель языка, машине нужен корпус — большой набор текстов. Чем больше, тем лучше. При этом желательно, чтобы его тексты по стилю не очень отличались от тех текстов, с которыми будет работать система. А если система должна работать с самыми разными текстами, то и корпус должен состоять из текстов самых разных. Множество всех словоформ корпуса обозначим греческой буквой ν. Это множество может быть очень большим, но оно конечное — число слов в нем ограничено. Поэтому множество ν можно задать обычным перечислением.

(4) ν = {этот, большой, дракон, мальчик, увидел, мальчика, …}

Предложения в языке — это цепочки слов из множества ν. В конце предложения для технических нужд необходим символ, обозначающий конец предложения. В письменных текстах это обычно точка. Но для машины лучше использовать какое-то другое обозначение. Например, написать слово STOP. Итак, предложения как цепочки слов могут быть, например, такие:

(5)

этот большой дракон увидел мальчика STOP

этот мальчик большой STOP

дракон этот этот STOP

дракон дракон дракон STOP

Каждому предложению, составленному из слов множества ν, необходимо приписать некоторую вероятность. Число предложений, которые можно построить из слов ν, не ограничено, поскольку длина предложения теоретически может быть какой угодно. Как посчитать эту вероятность? Есть очень простой и очевидный способ. Считаем общее число предложений в корпусе. Допустим, их 10000. Затем для каждого предложения подсчитываем, сколько раз оно встретилось в нашем корпусе. Например, предложение Мальчик увидел дракона встретилось 5 раз. Делим 5 на 10000, получаем вероятность. Для предложения Мальчик увидел дракона вероятность равна 0,0005. Для остальных предложений из корпуса вероятность высчитывается точно так же. А для отсутствующих в корпусе предложений устанавливается вероятность, равная нулю.

У этого способа есть один важный недостаток. Оно приписывает нулевую вероятность всем предложениям, которые в нем не встретились. Необходимо научиться оценивать вероятность для всех предложений, даже для тех, которые раньше нам никогда не встречались. Это можно сделать, анализируя фрагменты предложений — сочетания слов. При этом важно помнить, что вероятность появления слова зависит от того, какие слова появились перед ним. После слова манная скорее всего будет идти либо каша, либо крупа, либо запеканка. А если нам встретилось сочетание съешь манную, то за ним уже почти наверняка пойдет слово кашу, и ни на что другое надеяться не следует. Съешь манную запеканку в Интернете не встретилось. Иными словами, нас интересует вероятность слова xn, при условии, что перед ним идут слова x1, x2…xn-1. Это можно записать с помощью формулы условной вероятности:

(6) P (xn | x1, x2…xn-1)

Зная условную вероятность для каждого слова, можно посчитать вероятность любого предложения, состоящего из слов x1, x2, x3…. xn. Для этого можно просто перемножить условные вероятности каждого отдельного слова:

(7) P(x1, x2, x3…. xn) = P(x1) · P(x2 | x1) · P (x3 | x1, x2) …· P (xn | x1, x2…xn-1)

Эта несколько упрощенная запись означает, что вероятность цепочки слов x1, x2, x3…. xn равна вероятности появления первого слова x1, помноженной на условную вероятность появления второго слова x2 (при условии, что ему предшествует слово x1), помноженной на условную вероятность появления третьего слова x3 (при условии, что предыдущие два слова — это x1 и x2), и так далее, до условной вероятности последнего слова xn. Однако вычислять такую длинную цепочку очень сложно, особенно учитывая, что для этого нам потребуется слишком большой корпус исходных данных. Но даже и в нем могут не встретиться какие-то необходимые нам

длинные цепочки. Чтобы избежать этого, надо либо сильно увеличить корпус, стараясь включить в него все возможные сочетания, либо изобрести какой-то другой способ вычисления вероятности. Наиболее естественный альтернативный подход — опираться не на всю цепочку слов в предложении, а только на два-три предшествующих слова. Этот метод был разработан в конце XX века, задолго до появления компьютеров, русским математиком А. А. Марковым. Он предположил, что не так важно, учитывается ли вся длинная предшествующую цепочка или берется только один, два или три ближайших элемента — условная вероятность во всех этих случаях будет отличаться не очень сильно. Конечно, из этого обобщения бывают исключения. Но их мало, и для статистики они не очень важны. Так появились цепи Маркова — статистические модели, позволяющие сильно упростить вычисление условной вероятности. Цепь Маркова первого порядка самая грубая и самая простая. Она учитывает только одно предыдущее событие. Вероятность предложения на основе Марковской цепи первого порядка можно посчитать как вероятность первого слова, умноженную на вероятность второго слова при условии наличия первого слова, помноженную на вероятность третьего слова при условии наличия второго слова, и так далее, до вероятности последнего слова (которое у нас, кстати, всегда STOP) при наличии предпоследнего слова. Это можно записать с помощью формулы, которую мы, опять же, представим в несколько упрощенном виде, чтобы не утомлять читателя дополнительными объяснениями:

(8) $P(x1, x2, x3 \ldots xn) \approx P(x1) \cdot P(x2 \mid x1) \cdot P(x3 \mid x2) \ldots \cdot P(xn \mid xn\text{-}1)$

4 Задания для выполнения работы

– построить систему машинного перевода (русский → английский) на основе параллельного корпуса.

– ответить на контрольные вопросы

Данные для обучения: рекомендуется использовать новостной параллельный корпус News Commentary v11. Корпус содержит примерно 200 000 пар предложений. Вы можете использовать дополнительные/другие данные для обучения.

Инструменты: Moses или seq2seq.

Данные для тестирования: в качестве тестового набора вы получите несколько тысяч русских предложений (одно предложение в строке) и должны вернуть перевод этих предложений на английский (одно предложение в строке, с сохранением порядка предложений). Нельзя использовать существующие сторонние системы машинного перевода.

Оценка: для оценки используется среднее значение BLEU среди переводов предложений из тестового набора.

Методика выполнения

1. Для выполнения практического задания необходимо выбрать язык программирования Python. Данный язык является наиболее подходящим выбором для обработки естественного языка и машинного перевода текстов благодаря наличию большого количества библиотек, упрощающих выполнение задачи.

2. В данном задании использовалась библиотеки numpy, pickle, re.

3. Необходимо загрузить, нормализовать данные и токенизировать текст (рисунок 1).

```
# load doc into memory
def load_doc(filename):
    # open the file as read only
    file = open(filename, mode='rt', encoding='utf-8')
    # read all text
    text = file.read()
    # close the file
    file.close()
    return text

# split a loaded document into sentences
def to_pairs(doc):
    lines = doc.strip().split('\n')
    pairs = [line.split('\t') for line in  lines]
    return pairs

# clean a list of lines
def clean_pairs(lines):
    cleaned = list()
    # prepare regex for char filtering
    re_print = re.compile('[^%s]' % re.escape(string.printable))
    # prepare translation table for removing punctuation
    table = str.maketrans('', '', string.punctuation)
    for pair in lines:
        clean_pair = list()
        for line in pair:
            # normalize unicode characters
            line = normalize('NFD', line).encode('ascii', 'ignore')
            line = line.decode('UTF-8')
            # tokenize on white space
            line = line.split()
            # convert to lowercase
            line = [word.lower() for word in line]
            # remove punctuation from each token
            line = [word.translate(table) for word in line]
            # remove non-printable chars form each token
            line = [re_print.sub('', w) for w in line]
            # remove tokens with numbers in them
            line = [word for word in line if word.isalpha()]
            # store as string
            clean_pair.append(' '.join(line))
        cleaned.append(clean_pair)
    return array(cleaned)
```

Рисунок 1 – Загрузка данных

4. Далее необходимо построить модель и сохранить ее с помощью Pickle(рисунок 2).

```
# load dataset
filename = "../Data/for_translate.txt"
doc = load_doc(filename)
# split into english-russian pairs
pairs = to_pairs(doc)
# clean sentences
clean_pairs = clean_pairs(pairs)
# save clean pairs to file
save_clean_data(clean_pairs, 'english-russian.pkl')
```

Рисунок 2 – Построение модели

5. Теперь с помощью готовой модели необходимо перевести текст, используя полученную модель (рисунок 3).

```
with open('./Data/result.txt', 'w') as f:
    for item in items:
        f.write(item + '\n')
```

```
Saved: english-russian.pkl
Office SharePoint Designer 2007 reports of errors allow you to prevent the user about possible problems and correct them before
they affect other users.
We saw many cars, tents, families with children.
Meeting / Banquet Facilities, Nanny / Child Services, Laundry, Dry Cleaning, Currency Exchange, Souvenir Shop, Car Rental, Fax
/ Photocopying
The head of the IMF mission in Ukraine Jail Pazarbasioglu agrees to restore the balance of the balance in the global economy.
Since FMC is no longer responsible for the medical service of asylum applicants, you cannot continue to visit the same contact
person.
The MaxNet IPTV label should appear on the desktop, then run the program by pressing twice the MaxNet IPTV label.
Now there is also a school, and there is a modest memorial corner dedicated to an outstanding countryman.
Probably, this step will take you much more time than previously taken together.
The Armenian program was officially established in 1988. At the initiative of T.Guymjyan, with the material assistance of Ike a
nd Isabel Berberian.
Swedish neurologist Jonas Friessen believes that every adult person biologically on average fifteen and a half years.
Before we consider this aspect, you need to make a brief accession to the popular scientist source used by critics.
These kingdoms, most scientists recognized by Khmer, had close contacts with China and Thailand.
On a scale from 0 to 10, where 0 is the smallest level of knowledge, and 10 is the highest.
Sometimes I travel to those regions of the world where the death of humanity is so obvious that the heart is broken.
```

Рисунок 3 – Результат перевода

6. Полученный результат необходимо проверить средствами Stepik (рисунок 4).

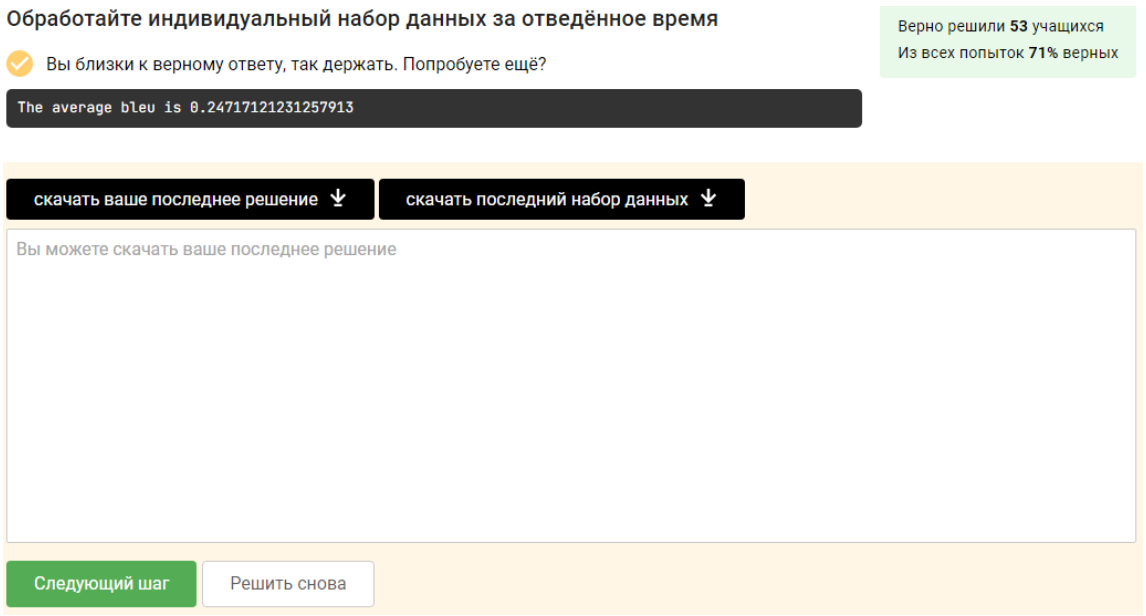

Рисунок 4 – Проверка результата

5 Варианты заданий

1. Если у разработчика в его проекте есть модуль, который подключается к FTP-серверу для загрузки некоторых файлов, то разработчик напишет метод подключения к FTP через URL-адрес FTP и использует учетные данные, такие как имя пользователя и пароль для успешного подключения. Если разработчик использует эти учетные данные в коде и жестко закодирует их в своем файле кода и развернет приложение, оно может работать нормально, работа выполнена! Теперь представьте, что через два месяца пароль для этого FTP-сайта изменен, и разработчик должен снова обновить этот пароль в вашем приложении, чтобы убедиться, что он не нарушает существующую функциональность FTP-соединения. Теперь в этом сценарии разработчик снова изменит код и повторно развернет его. Здесь разработчик должен для небольшого изменения конфигурации использовать последнюю версию кода, внести необходимые изменения, убедиться, что больше ничего не сломается, повторно развернуть приложение и затем протестировать его. И это может быть повторяющейся задачей через 2-3 месяца, когда какая-то конфигурация снова изменится. Что делать, если в приложении есть файл конфигурации, в котором есть пары ключ-значение “Пользователь”: “<имя пользователя>”, “Пароль”: “пароль”, и всякий раз, когда требуется какое-либо изменение, затрагивается только этот файл, и конфигурации обновляются, а не копаются в фактическом коде.

2. Исходя из опыта работы с .NET, мне было немного сложно понять, как разработчик может иметь файл конфигурации в приложении Python, который можно использовать для чтения значений настроек, и при этом не нужно даже прикасаться к коду для обновления или сохранения настроек. Конфигурационные файлы используются для хранения пар ключ-значение или

некоторой настраиваемой информации, которая может быть прочитана или доступна в коде и в определенный момент времени. Если эта конфигурация изменится, разработчики могут просто изменить конфигурацию в этом файле конфигурации и не беспокоиться об изменении кода, так как это может потребовать повторной компиляции кода и его развертывания. Не только это, но и использование конфигурационных файлов делает ваши настройки и код более удобными для повторного использования, а также позволяет хранить информацию о настройках централизованно и отдельно. Конечно, конфиденциальная информация, такая как пароли, секреты и сертификаты, должна храниться в большей безопасности и может находиться в облачных хранилищах. Но основные настройки, используемые в приложении, могут быть частью файла конфигурации.

3. Графики знаний существуют уже почти полвека – этот термин был впервые введен в обиход в 1972 году! Долгое время они просто томились в академическом мире, пока Google не анонсировала свой график знаний в 2012 году. С тех пор графики знаний эволюционировали довольно резко, и теперь пути назад нет.За последние 10 лет наблюдается стремительный рост машинного обучения и искусственного интеллекта . Благодаря своей способности внедрять интеллектуальные данные в данные и добавлять контекст, графики знаний используются для того, чтобы сделать МО и ИИ более надежными, заслуживающими доверия и объяснимыми. Вполне естественно, что эти две технологии сближаются, стимулируя рост и необходимость графиков знаний. Графики знаний преобразуют интеллектуальные данные в данные с помощью контекста и взаимосвязей. Графики знаний организуют и предоставляют единое место для поиска соответствующих данных и – что не менее важно – для интерпретации и принятия мер в соответствии с тем, что важно. Связанные данные, обогащенные смыслом, позволяют лучше отвечать на сложные запросы и находить наиболее эффективный путь к бесценной информации.

4. Инкапсуляция — это концепция объектно-ориентированного программирования, которая охватывает детали реализации класса. Почему бы не раскрыть детали реализации? Одна из хороших практик программирования заключается в том, что клиентский код должен иметь доступ только к открытым интерфейсам класса. Пока интерфейс остается прежним, клиентский код не нуждается в изменении в случае изменения реализации. Инкапсуляция также снижает сложность и упрощает процессы отладки. Кроме того, инкапсулированный класс помогает защитить данные и предотвратить неправильное использование, поскольку клиентский код не может получить доступ к защищенной части класса. В дополнение к деталям реализации, элементы данных, как правило, также являются участками кода, которые должны быть защищены. Однако может оказаться целесообразным предоставить общедоступный интерфейс, позволяющий клиентскому коду получать доступ к элементам данных в контексте класса. Этот тип интерфейса обычно называется функцией доступа. Функции доступа обычно бывают двух

видов: геттер и сеттер. Получатель — это метод, который вызывается, когда мы обращаемся к элементу данных для чтения. В отличие от геттера, сеттер — это метод, вызываемый для изменения элемента данных.

5. "Тестирование кода" обычно организуется в виде пути, который начинается с модульного тестирования, продолжается интеграцией/сквозным тестированием и заканчивается приемочным тестированием пользователей. В начале этого пути модульное тестирование выполняет две вещи — оно подтверждает, что "тестируемый код" работет правильно при подаче своих входных данных. После этого вы переходите к интеграционному тестированию, где подтверждаете, что "передача" работает правильно (предположительно, только после модульного тестирования "следующая вещь в цепочке обработки", чтобы подтвердить, что она выполняет свою работу правильно). Поскольку API-интерфейсы являются интерфейсом к цепочке обработки, легко спутать модульное тестирование API с интеграционным тестированием всей этой цепочки обработки. Эффективная стратегия тестирования API предотвращает это. Чтобы продемонстрировать, как эффективно управлять тестированием API, я собираюсь рассмотреть стратегию модульного тестирования API (с использованием [Telerik Test Studio для API), чтобы продемонстрировать, как выглядит эффективная стратегия тестирования API в реальных условиях.

6. Apache Spark - популярная система с открытым исходным кодом для обработки больших объемов данных. Большие данные могут быть большими — возможно, петабайтами. Он также может быть сложным, объединяющим несколько схем и источников. Это также может быть высокоскоростной поток данных. Все эти типы данных выигрывают от обширной параллельной обработки различных частей данных в памяти, чтобы свести к минимуму циклические перемещения в постоянное хранилище. Apache Spark облегчает это, автоматически масштабируя обработку и память для эффективного анализа больших наборов данных. Приложения для Apache Spark включают пакетную обработку фильтрации данных, агрегирование данных и преобразование данных в пригодные для использования наборы данных. Он также может использовать модели машинного обучения и анализа для обработки огромных объемов данных для создания моделей, поиска тенденций и прогнозирования будущих сценариев. Это также полезно для обработки данных в режиме реального времени, которая быстро обрабатывает и анализирует поток данных. Многие другие области применения Apache Spark ограничены только вашим воображением.

6 Контрольные вопросы:

1. Для фразы «Call me what instrument you will, though you can fret me, you cannot play upon me.» Упорядочите варианты перевода ниже по убыванию BLEU-2 (метрика на основе униграмм и биграмм).

a. назови меня каким угодно инструментом ты можешь меня расстроить, но не играть на мне

b. позвони мне какой инструмент ты будешь хотя ты можешь меня волновать, но ты не можешь играть на меня

c. назовите меня какой инструмент вы будете хотя вы можете раздражать меня все же вы не можете играть на меня

d. считай меня чем тебе угодно ты можешь мучить меня, но не играть мною

e. назовите мне какой инструмент вы хотите хотя можете меня беспокоить, но вы не можете играть на меня

f. позвони мне на каком инструменте вы будете хотя вы можете беспокоиться меня, но вы не можете играть на мне

Практическая работа №6 «Программирование и проектирование систем обработки естественных языков»

1 **Цель работы**: освоение практических навыков программирования и проектирования систем обработки естественных языков

2 **Порядок выполнения работы:**

– проработать краткие теоретические сведения;
– выполнить задания
– составить отчет о лабораторной работе и защитить его у преподавателя.
– ответить на контрольные вопросы

3 **Краткие теоретические сведения**

Обработка естественного языка (далее NLP — Natural language processing) — область, находящаяся на пересечении computer science, искусственного интеллекта и лингвистики. Цель заключается в обработке и “понимании” естественного языка для перевода текста и ответа на вопросы.

Человеческий язык полон двусмысленностей, которые невероятно затрудняют написание программного обеспечения, которое точно определяет предполагаемое значение текстовых или голосовых данных. Омонимы, омофоны, сарказм, идиомы, метафоры, исключения из грамматики и употребления, различия в структуре предложений — это лишь некоторые из нарушений человеческого языка, на изучение которых у людей уходят годы, но программисты должны научить приложения, основанные на естественном языке, распознавать и точно понимать с самого начала, если эти приложения будут полезны.

Несколько задач NLP разбивают человеческий текст и голосовые данные таким образом, чтобы помочь компьютеру понять, что он проглатывает. Некоторые из этих задач включают следующее:

Распознавание речи, также называемое преобразованием речи в текст, - это задача надежного преобразования голосовых данных в текстовые. Распознавание речи требуется для любого приложения, которое выполняет голосовые команды или отвечает на устные вопросы. Что делает распознавание речи особенно сложным, так это то, как люди говорят — быстро, невнятно произнося слова, с разным акцентом и интонацией, с разными акцентами и часто используя неправильную грамматику.

Пометка части речи, также называемая грамматической пометкой, представляет собой процесс определения части речи определенного слова или фрагмента текста на основе его использования и контекста. Часть речи определяет ‘сделать’ как глагол в "Я могу сделать бумажный самолетик" и как существительное в "Какой марки у вас автомобиль?"

Устранение неоднозначности смысла слова - это выбор значения слова с несколькими значениями посредством процесса семантического анализа, который определяет слово, которое имеет наибольший смысл в данном

контексте. Например, устранение неоднозначности смысла слова помогает отличить значение глагола "сделать" в "сделать оценку" (достичь) от "сделать ставку" (разместить).

Распознавание именованных сущностей, или NEM, идентифицирует слова или фразы как полезные сущности. NEM идентифицирует ‘Кентукки’ как местоположение или "Фред" как мужское имя.

Разрешение совместной ссылки - это задача определения того, относятся ли и когда два слова к одному и тому же объекту. Наиболее распространенным примером является определение лица или объекта, к которому относится определенное местоимение (например, "она" = "Мэри"), но оно также может включать определение метафоры или идиомы в тексте (например, пример, в котором "медведь" - это не животное, а большой волосатый человек).

Анализ настроений пытается извлечь из текста субъективные качества — отношение, эмоции, сарказм, замешательство, подозрение.

Генерацию естественного языка иногда описывают как противоположность распознаванию речи или преобразованию речи в текст; это задача перевода структурированной информации на человеческий язык.

С развитием голосовых интерфейсов и чат-ботов, NLP стала одной из самых важных технологий искусственного интеллекта. Но полное понимание и воспроизведение смысла языка — чрезвычайно сложная задача, так как человеческий язык имеет особенности:

- Человеческий язык — специально сконструированная система передачи смысла сказанного или написанного. Это не просто экзогенный сигнал, а осознанная передача информации. Кроме того, язык кодируется так, что даже маленькие дети могут быстро выучить его.
- Человеческий язык — дискретная, символьная или категориальная сигнальная система, обладающая надежностью.
- Категориальные символы языка кодируются как сигналы для общения по нескольким каналам: звук, жесты, письмо, изображения и так далее. При этом язык способен выражаться любым способом.

Где применяется NLP

Сегодня быстро растет количество полезных приложений в этой области:

— **поиск** (письменный или устный);
— показ подходящей **онлайн рекламы;**
— автоматический (или при содействии) **перевод;**
— **анализ настроений** для задач маркетинга;
— **распознавание речи и чат-боты,**
— голосовые **помощники** (автоматизированная помощь покупателю, заказ товаров и услуг).

Глубокое обучение в NLP

Существенная часть технологий NLP работает благодаря **глубокому обучению** (deep learning) — области машинного обучения, которая начала набирать обороты только в начале этого десятилетия по следующим причинам:

- Накоплены **большие объемы** тренировочных данных;

- Разработаны **вычислительные мощности**: многоядерные CPU и GPU;
- Созданы **новые модели и алгоритмы** с расширенными возможностями и улучшенной производительностью, с гибким обучением на промежуточных представлениях;
- Появились **обучающие методы с использованием контекста**, новые методы регуляризации и оптимизации.

Язык программирования Python предоставляет широкий спектр инструментов и библиотек для решения конкретных задач NLP. Многие из них можно найти в Инструментарии естественного языка, или NLTK, коллекции библиотек, программ и образовательных ресурсов с открытым исходным кодом для создания программ NLP.

NLTK включает библиотеки для многих задач NLP, перечисленных выше, плюс библиотеки для подзадач, таких как синтаксический анализ предложений, сегментация слов, вывод и лемматизация (методы обрезки слов до их корней) и токенизация (для разбиения фраз, предложений, абзацев и отрывков на токены, которые помогают компьютеру лучше понимать текст). Он также включает библиотеки для реализации таких возможностей, как семантическое мышление, способность делать логические выводы на основе фактов, извлеченных из текста.

Самые ранние приложения NLP были системами с ручным кодированием, основанными на правилах, которые могли выполнять определенные задачи NLP, но не могли легко масштабироваться для размещения, казалось бы, бесконечного потока исключений или увеличения объемов текстовых и голосовых данных.

Введите статистическое NLP, которое сочетает компьютерные алгоритмы с моделями машинного обучения и глубокого обучения для автоматического извлечения, классификации и маркировки элементов текстовых и голосовых данных, а затем присваивает статистическую вероятность каждому возможному значению этих элементов. Сегодня модели глубокого обучения и методы обучения, основанные на сверточных нейронных сетях (CNN) и рекуррентных нейронных сетях (RNN), позволяют системам NLP "учиться" во время работы и извлекать все более точный смысл из огромных объемов необработанных, неструктурированных и немаркированных наборов текстовых и голосовых данных.

Обработка естественного языка является движущей силой машинного интеллекта во многих современных приложениях реального мира. Вот несколько примеров:

Обнаружение спама: лучшие технологии обнаружения спама используют возможности классификации текста NLP для сканирования электронных писем на предмет текста, который часто указывает на спам или фишинг. Эти показатели могут включать чрезмерное использование финансовых терминов, характерную плохую грамматику, угрожающий язык, неуместную срочность, названия компаний с ошибками и многое другое. Обнаружение спама - одна из немногих проблем NLP, которые эксперты считают "в основном решенными"

(хотя вы можете возразить, что это не соответствует вашему опыту работы с электронной почтой).

Машинный перевод: Google Translate является примером широко доступной технологии NLP в действии. По-настоящему полезный машинный перевод включает в себя нечто большее, чем просто замену слов одного языка словами другого. Эффективный перевод должен точно отражать значение и тон языка ввода и переводить его в текст с тем же значением и желаемым воздействием на языке вывода. Инструменты машинного перевода значительно продвинулись в плане точности. Отличный способ протестировать любой инструмент машинного перевода - перевести текст на один язык, а затем вернуться к оригиналу. Часто цитируемый классический пример: не так давно при переводе “Дух силен, но плоть слаба” с английского на русский и обратно получилось “Водка хорошая, но мясо гнилое”. Сегодня результатом является “Дух желает, но плоть слаба”, что не идеально, но внушает гораздо больше доверия в переводе с английского на русский.

Виртуальные агенты и чат-боты: Виртуальные агенты, такие как Siri от Apple и Alexa от Amazon, используют распознавание речи для распознавания шаблонов в голосовых командах и генерации естественного языка, чтобы отвечать соответствующими действиями или полезными комментариями. Чат-боты выполняют ту же магию в ответ на введенные текстовые записи. Лучшие из них также учатся распознавать контекстуальные подсказки о запросах людей и использовать их для предоставления еще лучших ответов или вариантов с течением времени. Следующее усовершенствование для этих приложений — ответы на вопросы, возможность отвечать на наши вопросы — ожидаемые или нет - соответствующими и полезными ответами своими словами.

Анализ настроений в социальных сетях: NLP стало важным бизнес-инструментом для выявления скрытых данных из каналов социальных сетей. Анализ настроений позволяет анализировать язык, используемый в сообщениях в социальных сетях, ответах, обзорах и т. Д., Чтобы выявить отношение и эмоции в ответ на продукты, рекламные акции и события – информацию, которую компании могут использовать в дизайне продуктов, рекламных кампаниях и многом другом.

Обобщение текста: Обобщение текста использует методы NLP для обработки огромных объемов цифрового текста и создания резюме и резюме для индексов, исследовательских баз данных или занятых читателей, у которых нет времени читать полный текст. Лучшие приложения для обобщения текста используют семантические рассуждения и генерацию естественного языка (NLG) для добавления полезного контекста и выводов в резюме.

Обзор NLP Architect

Команда NLP-исследователей и разработчиков из Intel AI Lab занимается изучением актуальных архитектур глубоких нейросетей и методов обработки и понимания текста. Результатом их работы стал набор инструментов, интересных как с теоретической, так и с прикладной точки зрения.

Вот что есть в текущей версии NLP Architect:

— Модели, извлекающие лингвистические характеристики текста: например, синтаксический анализатор (BIST) и алгоритм для извлечения именных групп (noun phrases);
— State-of-the-art модели для понимания языка: например, определение намерения пользователя (intent extraction), распознавание именованных сущностей (named entity recognition, NER);
— Модули для семантического анализа: например, коллокации, наиболее вероятный смысл слова, векторные представления именованных групп (NP2V);
— Строительные блоки для создания разговорного интеллекта: например, основа для создания чатботов, включающая в себя систему поддержания диалога, анализатор предложений (sequence chunker), система для определения намерения пользователя;
— Примеры применения глубоких end-to-end нейросетей с новой архитектурой: например, вопросно-ответные системы, системы понимания текста (machine reading comprehension).

Для всех вышеупомянутых моделей есть примеры процессов обучения и предсказания. Более того, команда Intel добавила скрипты для решения типичных задач, возникающих при внедрении таких моделей—конвейеры для обработки данных и утилиты, часто применяющиеся в NLP.

Библиотека состоит из отдельных модулей, что упрощает интеграцию. Общий вид фреймворка NLP Architect изображён на схеме ниже (рис.6.1).

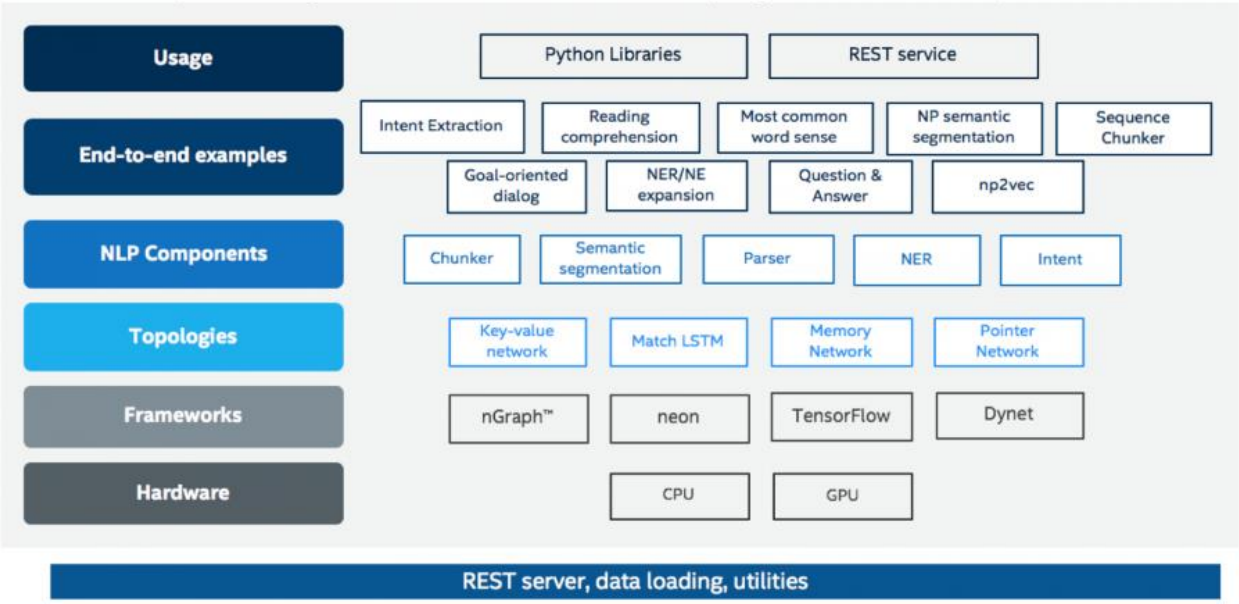

Рисунок 6.1 Обобщенная схема фреймворка NLP Architect

Компоненты для NLP/NLU

Рассмотрим некоторые модули. Анализатор предложений. Анализ предложений (sequence chunking)—одна из базовых задач обработки текста,

которая заключается в разделении предложения на синтаксически связанные части. Например, предложение

“Маленькая Саша шла по шоссе”

можно разделить на четыре части: именные группы “Маленькая Саша” и “шоссе”, глагольную группу “шла” и предложную группу “по”.

Анализатор предложений из NLP Architect умеет строить подходящую архитектуру нейросети для разных типов входных данных: токенов, меток частей речи, векторных представлений, символьных признаков.

Семантический сегментатор именных групп. Именная группа (noun phrase) состоит из главного члена—существительного или местоимения—и нескольких зависимых уточняющих членов. Упрощая, можно разделить именные группы на два типа:

- с описательной структурой: зависимые члены не влияют существенно на семантику главного члена (например, “морская вода”);
- с коллокационной структурой: зависимые члены существенно изменяют смысл главного члена (например, “морская свинка”).

Для определения типа именной группы обучается многослойный персептрон. Эта модель используется в алгоритме семантической сегментации предложений. В результате её работы именные группы первого типа распадаются на несколько семантических элементов, а именные группы второго типа остаются едиными.

Синтаксический анализатор выполняет грамматический анализ предложений, рассматривая отношения между словами в предложениях и выделяя такие вещи, как прямые дополнения и сказуемые. В NLP Architect входит парсер зависимостей, основанный на графах, который использует BiLSTM для извлечения признаков..

Распознаватель именованных сущностей (NER) выделяет в тексте определённые слова или сочетания слов, относящиеся к некоторому интересующему нас классу. К примерам сущностей относятся имена, числа, места, валюты, даты, организации. Иногда сущности можно довольно легко выделить с помощью таких признаков, как форма слов, наличие слова в определённом словаре, часть речи. Однако довольно часто эти признаки нам не известны или даже не существуют. В таких случаях для того, чтобы определить, является ли слово или словосочетание сущностью, необходимо анализировать его контекст.

Модель для NER в NLP Architect основана на двунаправленной LSTM-сети и CRF-классификаторе 9 РИС 6.2).

NLP Architect—открытая и гибкая библиотека с алгоритмами для обработки текста, которая даёт возможность для взаимодействия разработчиков со всего мира. Команда Intel продолжает добавлять в библиотеку результаты своих исследований, чтобы любой мог воспользоваться тем, что они сделали и улучшили.

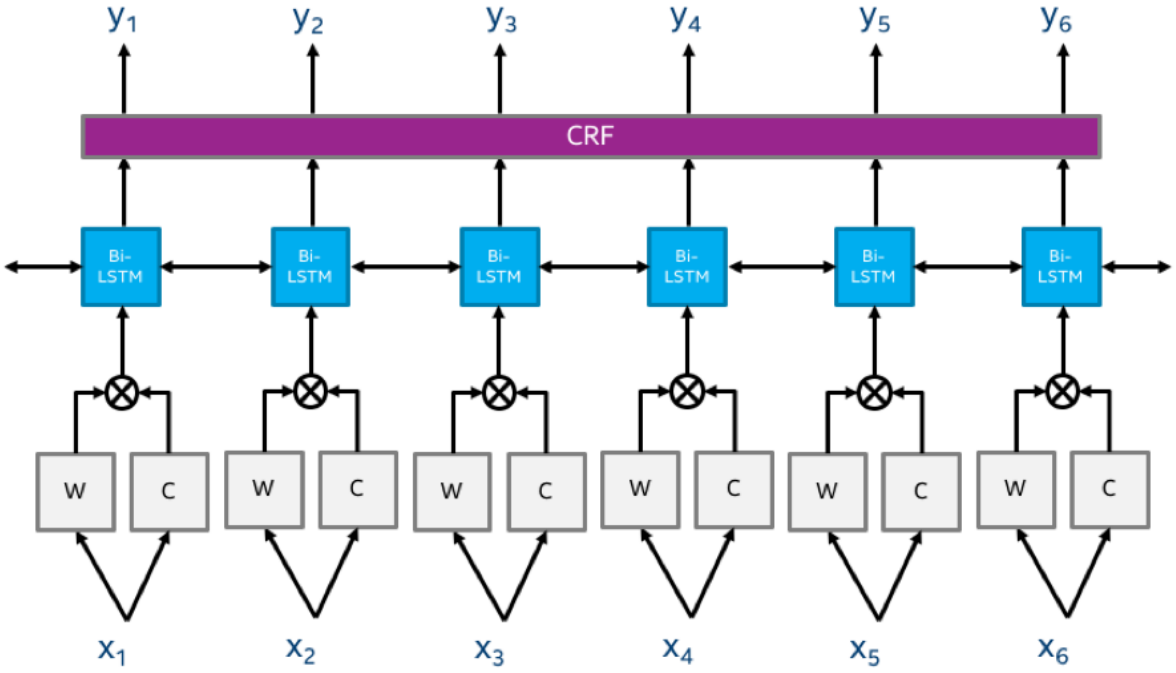

Рисунок 6.2 Обобщенная схема модель для NER в NLP Architect основаной на двунаправленной LSTM-сети

Для начала работы достаточно скачать код с репозитория на Гитхабе и выполнить инструкции по установке. Здесь можно найти исчерпывающую документацию для всех основных модулей и готовых моделей.

В будущих релизах Intel AI Lab планирует продемонстрировать преимущества создания алгоритмов анализа текста с помощью новейших технологий глубокого обучения, и включить в библиотеку методы для извлечения тональности текста, анализа тематик и трендов, расширения специализированных лексиконов и извлечения отношений. Кроме того, специалисты из Intel исследуют методы обучения без учителя и частичного обучения, с помощью которых можно создать новые интерпретируемые модели для понимания и анализа текста, способные адаптироваться к новым областям знания.

Методы

Векторное представление (text embeddings)

В традиционном NLP слова рассматриваются как дискретные символы, которые далее представляются в виде one-hot векторов. Проблема со словами — дискретными символами — отсутствие определения схожести для one-hot векторов. Поэтому альтернатива — обучиться кодировать схожесть в сами векторы.

Векторное представление — метод представления строк, как векторов со значениями. Строится плотный вектор (dense vector) для каждого слова так, чтобы встречающиеся в схожих контекстах слова имели схожие вектора. Векторное представление считается стартовой точкой для большинства NLP задач и делает глубокое обучение эффективным на маленьких датасетах. Техники векторных представлений Word2vec и **GloVe**, созданных Google

(Mikolov) Stanford (Pennington, Socher, Manning) соответственно, пользуются популярностью и часто используются для задач NLP. Давайте рассмотрим эти техники.

Word2vec принимает большой корпус (corpus) текста, в котором каждое слово в фиксированном словаре представлено в виде вектор (рис6.3)а. Далее алгоритм пробегает по каждой позиции *t* в тексте, которая представляет собой центральное слово *c* и контекстное слово *o*. Далее используется схожесть векторов слов для *c* и *o*, чтобы рассчитать вероятность о при заданном *c* (или наоборот), и продолжается регулировка вектор слов для максимизации этой вероятности.

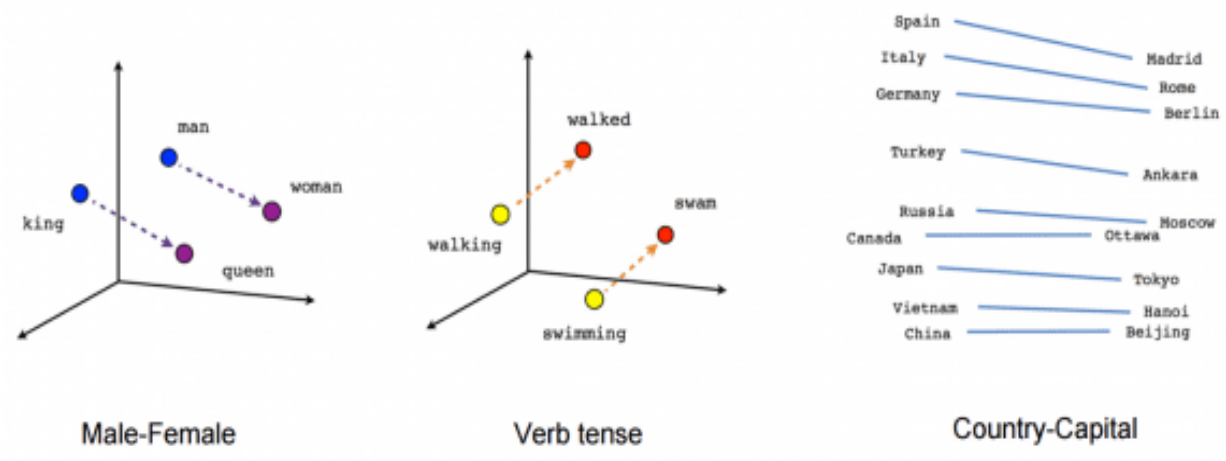

Рисунок 6.3- Схема векторного представления строк в динамике

Для достижения лучшего результата Word2vec из датасета удаляются бесполезные слова (или слова с большой частотой появления, в английском языке — **a,the,of,then**). Это поможет улучшить точность модели и сократить время на тренировку. Кроме того, используется отрицательная выборка (negative sampling) для каждого входа, обновляя веса для всех правильных меток, но только на небольшом числе некорректных меток.

Word2vec представлен в 2 вариациях моделей:

1. **Skip-Gram**: рассматривается контекстное окно, содержащее *k* последовательных слов. Далее пропускается одно слово и обучается нейронная сеть, содержащая все слова, кроме пропущенного, которое алгоритм пытается предсказать. Следовательно, если 2 слова периодически делят схожий контекст в корпусе, эти слова будут иметь близкие векторы (рис.6.4).
2. **Continuous Bag of Words**: берется много предложений в корпусе. Каждый раз, когда алгоритм видим слово, берется соседнее слово. Далее на вход нейросети подается контекстные слова и предсказываем слово в центре этого контекста. В случае тысяч таких контекстных слов и центрального слова, получаем один экземпляр датасета для нашей нейросети. Нейросеть тренируется и ,наконец, выход закодированного скрытого слоя представляет вложение (embedding) для определенного слова. То же происходит, если нейросеть тренируется на большом числе

предложений и словам в схожем контексте приписываются схожие вектора.

Единственная жалоба на Skip-Gram и CBOW — принадлежность к классу window-based моделей, для которых характерна низкая эффективность использования статистики совпадений в корпусе, что приводит к неоптимальным результатам.

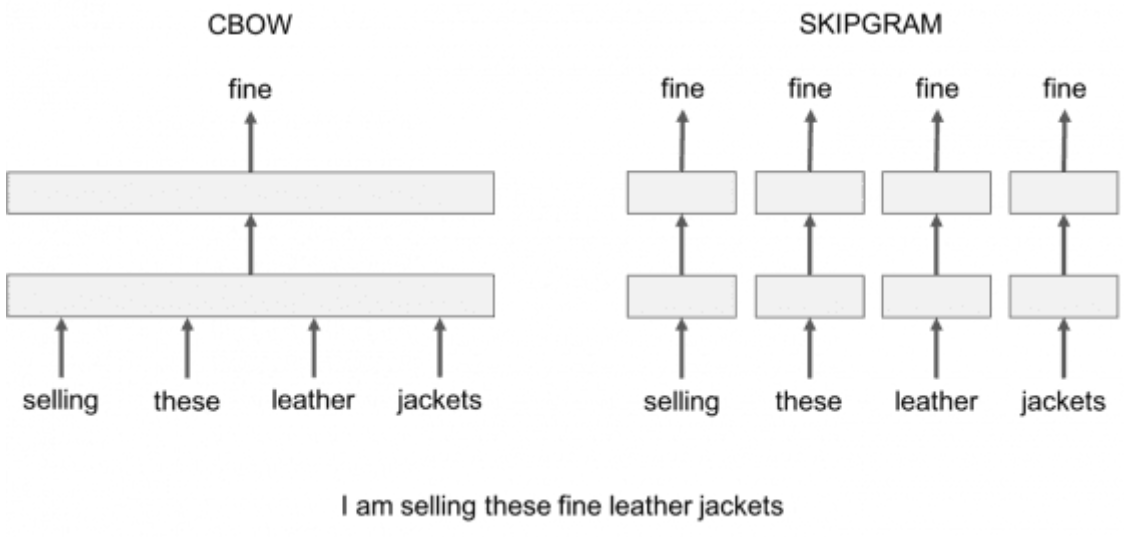

Рисунок 6.4- Схема отбора слов

GloVe стремится решить эту проблему захватом значения одного word embedding со структурой всего обозримого корпуса. Чтобы сделать это, модель ищет глобальные совпадения числа слов и использует достаточно статистики, минимизирует среднеквадратичное отклонение, выдает пространство вектора слова с разумной субструктурой. Такая схема в достаточной степени позволяет отождествлять схожесть слова с векторным расстоянием (рис.6.5).

Латентное размещение Дирихле (LDA, от англ. Latent Dirichlet allocation) — применяемая в машинном обучении и информационном поиске порождающая модель, позволяющая объяснять результаты наблюдений с помощью неявных групп, благодаря чему возможно выявление причин сходства некоторых частей данных.

Например, если наблюдениями являются слова, собранные в документы, утверждается, что каждый документ представляет собой смесь небольшого количества тем и что появление каждого слова связано с одной из тем документа. LDA является одним из методов тематического моделирования и впервые был представлен в качестве графовой модели для обнаружения тематик Дэвидом Блеем, Эндрю Ыном и Майклом Джорданом в 2003 году.

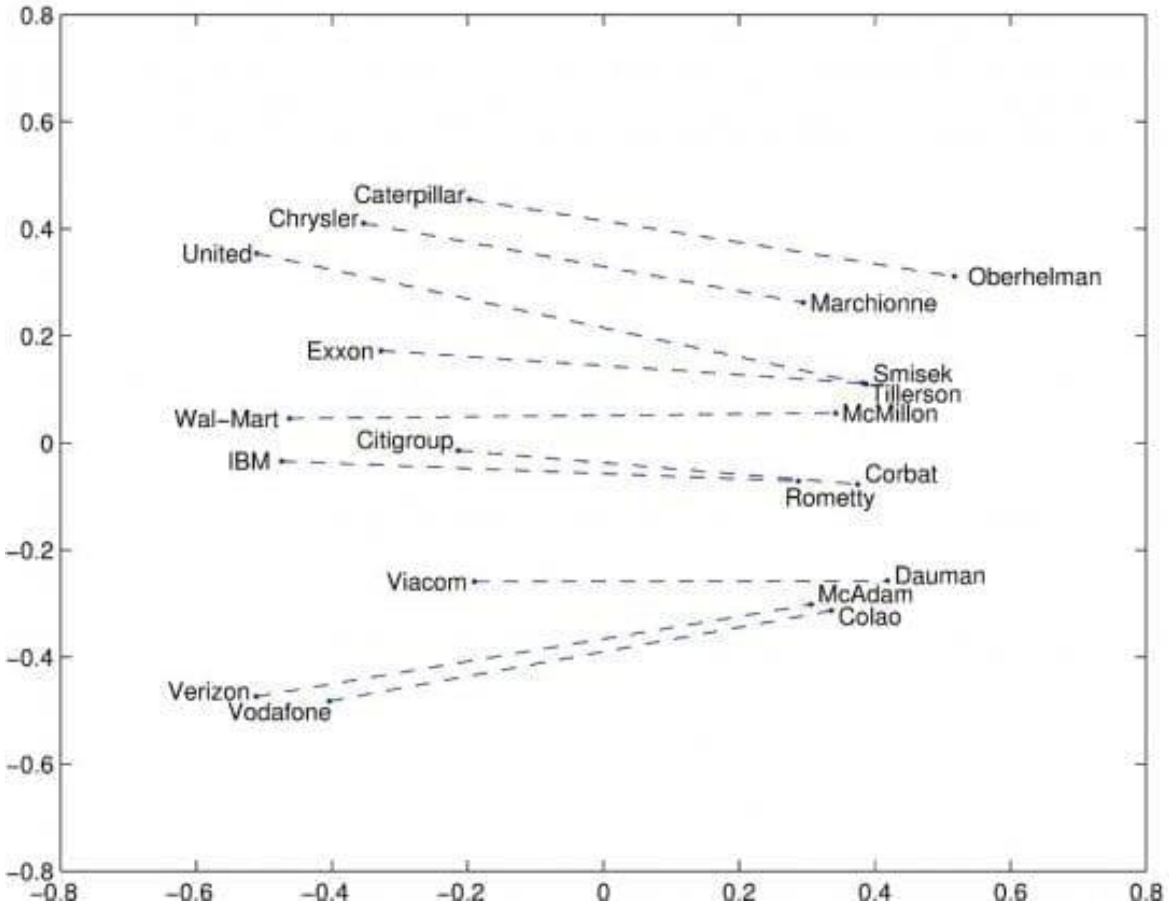

Рисунок 6.5- Схема поиска глобальных совпадений

Помимо этих двух моделей, нашли применение много недавно разработанных технологий: FastText, Poincare Embeddings, sense2vec, Skip-Thought, Adaptive Skip-Gram.

В LDA каждый документ может рассматриваться как набор различных тематик. Подобный подход схож с вероятностным латентно-семантическим анализом (pLSA) с той разницей, что в LDA предполагается, что распределение тематик имеет в качестве априори распределения Дирихле. На практике в результате получается более корректный набор тематик.

К примеру, модель может иметь тематики классифицируемые как «относящиеся к кошкам» и «относящиеся к собакам», тематика обладает вероятностями генерировать различные слова, такие как «мяу», «молоко» или «котёнок», которые можно было бы классифицировать как «относящиеся к кошкам», а слова, не обладающие особой значимостью (к примеру, служебные слова), будут обладать примерно равной вероятностью в различных тематиках.

Рассмотрим использование компонента скрытого выделения дирихле метода в конструкторе Машинное обучение Azure для группирования неклассифицированного текста в категории.

Скрытое распределение Дирихле (LDA) часто используется при обработке естественного языка для поиска похожих текстов. Другой распространенный термин — тематическое моделирование.

Этот компонент принимает столбец текста и создает следующие выходные данные:

Исходный текст вместе с оценкой по каждой категории

Матрица характеристик, содержащая извлеченные термины и коэффициенты для каждой категории

Преобразование, которое вы можете сохранить и повторно применить к новому тексту, используемому в качестве ввода

Этот компонент использует библиотеку scikit-учиться. Дополнительные сведения о scikit-learn см. в репозитории GitHub, который включает учебные пособия и объяснение алгоритма.

LDA обычно не является методом классификации. Но он использует генеративный подход, поэтому вам не нужно предоставлять известные метки классов, а затем делать выводы о шаблонах. Вместо этого алгоритм генерирует вероятностную модель, которая используется для определения групп тем. Вы можете использовать вероятностную модель для классификации существующих обучающих примеров или новых случаев, которые вы предоставляете модели в качестве входных данных.

Вы можете предпочесть генеративную модель, потому что она позволяет избежать серьезных предположений о взаимосвязи между текстом и категориями. Он использует только распределение слов для математического моделирования тем.

Реализация в этом компоненте основана на библиотеке scikit-учиться для Lda.

Как настроить латентное размещение Дирихле?

Для этого компонента требуется набор данных, содержащий столбец текста, либо необработанный, либо предварительно обработанный.

Добавьте в конвейер компонент скрытого выделения Дирихле метода .

В качестве входных данных для компонента укажите набор данных, содержащий один или несколько текстовых столбцов.

В поле Целевые столбцы выберите один или несколько столбцов, содержащих текст для анализа.

Вы можете выбрать несколько столбцов, но они должны быть строкового типа.

Поскольку LDA создает большую матрицу функций из текста, вы обычно анализируете один текстовый столбец.

В поле Число тем для моделирования введите целое число от 1 до 1000, которое указывает, сколько категорий или тем вы хотите получить из введенного текста.

По умолчанию создано 5 тем.

Для N-граммов укажите максимальную длину N-граммов, сгенерированных во время хэширования.

По умолчанию — 2, что означает, что генерируются и биграммы, и униграммы.

Выберите параметр Нормализовать, чтобы преобразовать выходные значения в вероятности.

Вместо того чтобы представлять преобразованные значения в виде целых чисел, значения в выходных данных и наборе классов объектов будут преобразованы следующим образом:

Значения в наборе данных будут представлены как вероятность, где P(topic|document).

Зачения в матрице тематических характеристик будут представлены как вероятность, где P(word|topic).

Примечание

В конструкторе машинного обучения Azure библиотека scikit-learn больше не поддерживает ненормализованный вывод doc_topic_distr из версии 0.19. В этом компоненте параметр нормализации может применяться только к выходным данным в матрице разделов компонентов . Вывод преобразованного набора данных всегда нормализуется.

Выберите параметр Показать все параметры, а затем установите для него значение TRUE, если вы хотите установить следующие дополнительные параметры.

Эти параметры относятся к реализации LDA в scikit-learn. В scikit-learn есть несколько хороших руководств по LDA, а также официальный документ scikit-learn.

Rho параметр. Обеспечьте априорную вероятность разреженности тематических распределений. Этот параметр соответствует параметру topic_word_prior в sklearn. Используйте значение 1, если вы ожидаете, что распределение слов будет ровным; то есть все слова считаются равновероятными. Если вы думаете, что большинство слов отображается редко, вы можете установить меньшее значение.

Альфа-параметр. Укажите априорную вероятность разреженности весов тем для отдельных документов. Этот параметр соответствует параметру doc_topic_prior в sklearn.

Примерное количество документов. Введите число, которое представляет собой наиболее точную оценку количества документов (строк), которые будут обработаны. Этот параметр позволяет компоненту выделить хэш-таблицу достаточного размера. Он соответствует параметру total_samples в scikit-learn.

Размер партии. Введите число, указывающее, сколько строк включать в каждый пакет текста, отправляемый в модель LDA. Этот параметр соответствует параметру batch_size в scikit-learn.

Начальное значение итерации, используемое в расписании обновления обучения. Укажите начальное значение, которое снижает скорость обучения для ранних итераций онлайн-обучения. Этот параметр соответствует параметру learning_offset в scikit-learn.

Мощность, применяемая к итерации во время обновлений. Укажите уровень мощности, применяемой к счетчику итераций, чтобы контролировать

скорость обучения во время онлайн-обновлений. Этот параметр соответствует параметру learning_decay в scikit-learn.

Количество проходов по данным. Укажите максимальное количество циклов, которое алгоритм будет перебирать данные. Этот параметр соответствует параметру max_iter в scikit-learn.

Выберите опцию Создать словарь n-грамм или Создать словарь n-грамм до LDA, если вы хотите создать список n-грамм на начальном этапе перед классификацией текста.

Если вы создадите исходный словарь заранее, вы можете позже использовать словарь при просмотре модели. Способность отображать результаты в виде текста, а не числовых индексов, как правило, легче интерпретировать. Однако сохранение словаря займет больше времени и потребует дополнительного места для хранения.

В поле Максимальный размер словаря ngram введите общее количество строк, которые можно создать в словаре n-грамм.

Этот параметр полезен для управления размером словаря. Но если число n-граммы во входных данных превышает этот размер, могут возникать конфликты.

Отправьте конвейер. Компонент LDA использует алгоритм Байеса теорема, чтобы определить, какие темы могут быть связаны с отдельными словами. Слова не связаны исключительно с какими-либо темами или группами. Вместо этого каждая n-грамма имеет изученную вероятность быть связанной с любым из обнаруженных классов.

Результаты

Компонент имеет два выхода:

Преобразованный набор данных: эти выходные данные содержат входной текст, указанное количество обнаруженных категорий и оценки для каждого примера текста для каждой категории.

Матрица тематических тем: крайний левый столбец содержит извлеченный текстовый объект. Столбец для каждой категории содержит оценку данной функции в этой категории.

Преобразование LDA

Этот компонент также выводит Преобразование Lda , которое применяет Lda к набору данных.

Вы можете сохранить это преобразование и повторно использовать его для других наборов данных. Этот метод может быть полезен, если вы тренировались на большом корпусе и хотите повторно использовать коэффициенты или категории.

Чтобы повторно использовать это преобразование, щелкните значок зарегистрировать набор данных на правой панели компонента скрытого выделения Дирихле метода, чтобы компонент оставался в категории наборы данных в списке компонентов. Затем можно подключить этот компонент к компоненту " Применить преобразование ", чтобы повторно использовать это преобразование.

Уточнение модели или результатов LDA

Как правило, невозможно создать единую модель LDA, которая удовлетворяла бы всем требованиям. Даже модель, разработанная для одной задачи, может потребовать множества итераций для повышения точности. Мы рекомендуем вам попробовать все эти методы для улучшения вашей модели:

Изменение параметров модели

Использование визуализации для понимания результатов

Получение обратной связи от профильных экспертов, чтобы определить, полезны ли созданные темы

Качественные меры также могут быть полезны для оценки результатов. Чтобы оценить результаты тематического моделирования, рассмотрите:

Достоверность. Действительно ли похожие предметы похожи?

Разнообразие. Может ли модель различать похожие элементы, когда это требуется для решения бизнес-задачи?

масштабируемость; Работает ли он с широким спектром текстовых категорий или только с узким целевым доменом?

Вы часто можете повысить точность моделей, основанных на LDA, используя обработку естественного языка для очистки, резюмирования и упрощения или категоризации текста. Например, следующие методы, все из которых поддерживаются в Машинном обучении Azure, могут повысить точность классификации:

— Удаление стоп-слов
— Нормализация регистра
— Лемматизация или извлечение корней
— аспознавание именованных сущностей

В дизайнере вы также можете использовать библиотеки R или Python для обработки текста: Execute R Script, Execute Python Script.

Сведения о реализации

По умолчанию распределения выходных данных для преобразованного набора данных и тематической матрицы нормализованы как вероятности:

Преобразованный набор данных нормализуется как условная вероятность тематики данного документа. В этом случае сумма каждой строки равна 1.

Матрица тематика-тема нормализована как условная вероятность слов, заданных в теме. В этом случае сумма каждого столбца равна 1.

Иногда компонент может вернуть пустой раздел. Чаще всего причиной является псевдослучайная инициализация алгоритма. Если это произойдет, вы можете попробовать изменить соответствующие параметры. Например, измените максимальный размер словаря N-граммов или количество битов, используемых для хэширования функций.

LDA и тематическое моделирование

Латентное распределение Дирихле часто используется для моделирования тем на основе содержания, что в основном означает изучение

категорий из неклассифицированного текста. В тематическом моделировании на основе содержания тема — это распределение слов.

Например, предположим, что вы предоставили корпус отзывов клиентов, включающий множество продуктов. Текст отзывов, которые были отправлены клиентами с течением времени, содержит множество терминов, некоторые из которых используются в нескольких темах.

Тема, которую определяет процесс LDA, может представлять собой обзоры отдельного продукта или группу обзоров продукта. Для LDA сама тема — это просто распределение вероятностей во времени для набора слов.

Условия редко бывают эксклюзивными для какого-либо одного продукта. Они могут относиться к другим продуктам или быть общими терминами, относящимися ко всему ("отлично", "ужасно"). Другие термины могут быть шумовыми словами. Однако метод LDA не пытается охватить все слова во вселенной или понять, как слова связаны между собой, за исключением вероятностей совместного появления. Он может группировать только слова, которые используются в целевом домене.

После того как индексы терминов вычислены, мера сходства на основе расстояния сравнивает отдельные строки текста, чтобы определить, похожи ли два фрагмента текста. Например, вы можете обнаружить, что у продукта есть несколько сильно коррелированных названий. Или вы можете обнаружить, что с определенным продуктом обычно связаны резко отрицательные термины. Вы можете использовать меру сходства как для определения связанных терминов, так и для создания рекомендаций.

Morphotactics

В основном это модель упорядочения морфем. В другом смысле модель, объясняющая, какие классы морфем могут следовать за другими классами морфем внутри слова. Например, морфотаксический факт заключается в том, что морфема английского множественного числа всегда следует за существительным, а не предшествует ему.

Орфографические правила

Эти правила правописания используются для моделирования изменений, происходящих в слове. Например, правило преобразования у в слова, например, город + с = города, а не города.

Обобщенный алгоритм обработки текста на естественном языке может быть представлен схематически (рис. 6.6).

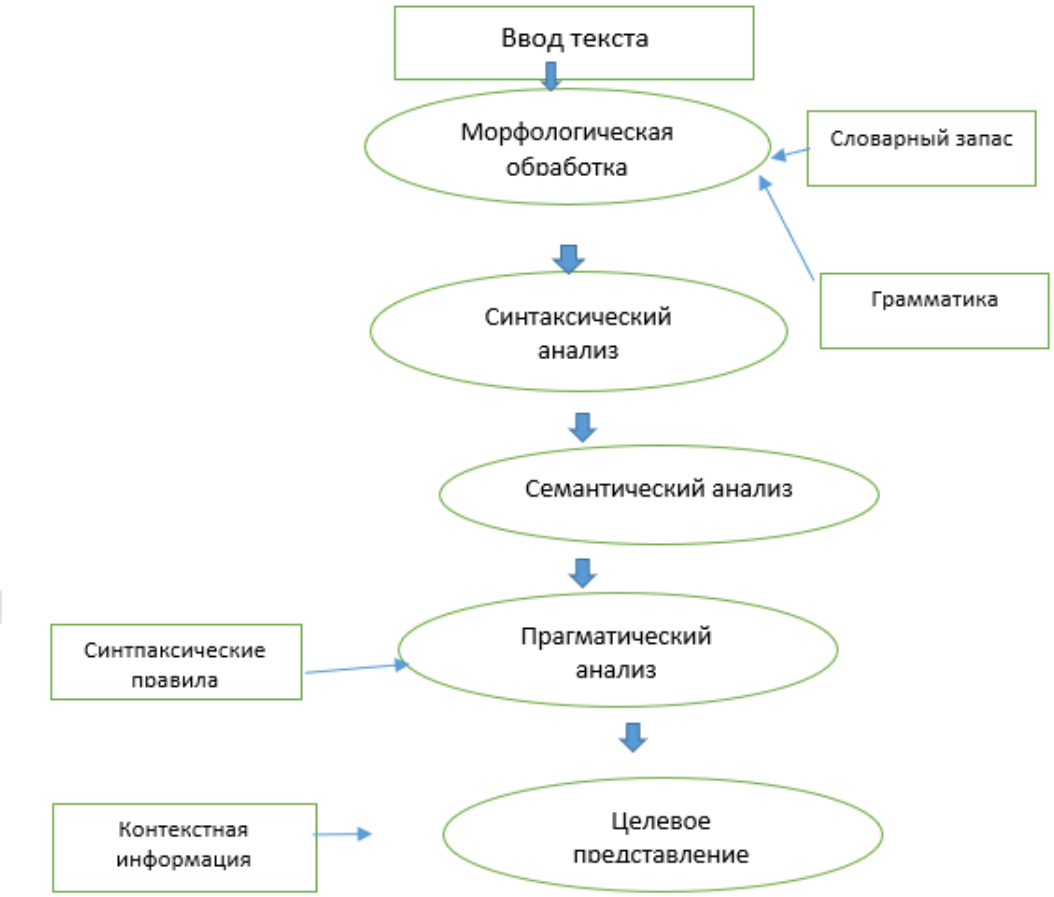

Рисунок 6.6 — Обобщенный алгоритм обработки текста на естественном языке

Обработка естественного языка — синтаксический анализ

Синтаксический анализ, синтаксический анализ или синтаксический анализ — это третья фаза NLP. Цель этого этапа — нарисовать точное значение, или вы можете сказать значение словаря из текста. Синтаксический анализ проверяет текст на предмет значимости по сравнению с правилами формальной грамматики. Например, предложение типа «горячее мороженое» будет отклонено семантическим анализатором.

В этом смысле синтаксический анализ или синтаксический анализ могут быть определены как процесс анализа строк символов на естественном языке в соответствии с правилами формальной грамматики. Происхождение слова ***«парсинг»*** происходит от латинского слова ***«парс»,*** что означает ***«часть»*** .

Концепция парсера

. Он также строит структуру данных, как правило, в форме дерева разбора или абстрактного синтаксического дерева или другой иерархической структур (Рис 6.7).

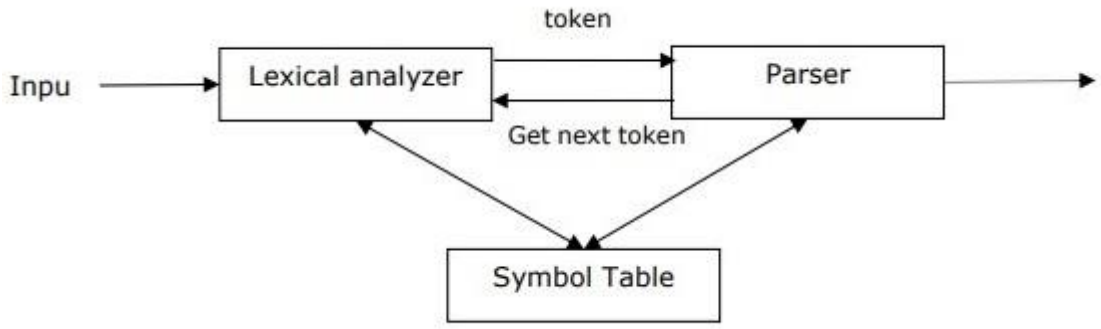

Рисунок 6.7 Обобщенная схема концепции парсера

Используется для реализации задачи разбора. Он может быть определен как программный компонент, предназначенный для сбора входных данных (текста) и обеспечения структурного представления входных данных после проверки правильности синтаксиса в соответствии с формальной грамматикой

Инструментарий

Есть много языков программирования общего назначения. Мы можем ранжировать их по популярности, но выделить лучший по общим характеристикам невозможно. Если бы был такой идеальный язык программирования, то все разработчики использовали бы его. Поскольку такого языка нет, каждому из нас приходится выбирать подходящий язык программирования в соответствии с нашими потребностями.

Делайте свой выбор между Bash, Python и JavaScript при написании Shell-сценариев, основываясь на следующих фактах и условиях:

— Если вам нужно часто инициировать процессы и писать небольшой переносимый Shell-сценарий для Unix или Unix-подобных операционных систем, Bash, несомненно, будет хорошим выбором. Например, я написал сценарий Bash для сборки двоичных файлов проекта с открытым исходным кодом, основанного на пользовательской архитектуре.

— Если вам нужно написать кроссплатформенный Shell-скрипт для обработки некоторых данных и выполнения определенных команд, можете выбрать Python. Например, в проекте Electron есть несколько сценариев Python для обработки и загрузки файлов. Однако не стоит ожидать высокой производительности от Shell-сценариев на базе Python.

— JavaScript отлично подходит для тех же сценариев, что и Python. Однако, в отличие от Python, JavaScript имеет некоторые дополнительные преимущества. JavaScript быстр, изначально поддерживает JSON и имеет

впечатляющие встроенные функции. Я нашел несколько сценариев Shell на базе Javascript в каталоге сценариев репозитория React Native.

Одним из главных языков программирования в области обработки текстов на естественном языке является Python. Скачать его можно с официального сайта http://www.python.org .

Чаще всего в задачах для бизнеса исходные данные предоставляются в формате .xlsx или .xlsm, однако многие предпочитают формат .csv (файлы, в которых каждая строка представлена полями, разделенными каким-либо знаком — обычно запятой или точкой с запятой). Поэтому для начала необходимо потренироваться в загрузке данных.

Чтение данных из файла на ЯП Python можно реализовать как:

```
from pandas import read_csv, DataFrame, Series
data = read_csv('Kaggle_Titanic/Data/train.csv')
```

Вывод данных на график (один из возможных вариантов):

```
fig, axes = plt.subplots(ncols=2)
data.pivot_table('PassengerId', ['SibSp'], 'Survived',
'count').plot(ax=axes[0], title='SibSp')
data.pivot_table('PassengerId', ['Parch'], 'Survived',
'count').plot(ax=axes[1], title='Parch')
```

OpenPyXL — это библиотека для работы исключительно с Excel-файлами, такими как .xlsx, .xlsm, .xltx, .xltm для версий Excel от 2010 года и новее. OpenPyXL содержит инструменты для чтения, записи и обработки данных указанных форматов, а также для построения графиков.

Библиотеки:— xlrd – дает возможность читать файлы Excel

— xlwt – создание и заполнение файлов Excel

— xlutils – набор утилит для расширения возможности предыдущих двух библиотек

— pyExcelerator – также дает возможность работать с файлами Excel, но давно не обновлялась.

Пример кода для работы с Excel (необходимо проверять корректность версий библиотек и MS Office!)[1]

```
import xlrd
rb = xlrd.open_workbook('d:/final.xls',formatting_info=True)
sheet = rb.sheet_by_index(0)
for rownum in range(sheet.nrows):
row = sheet.row_values(rownum)
for c_el in row:
print c_el
```

Создание нового файла:

```
import xlwt
from datetime import datetime

font0 = xlwt.Font()
```

[1] Подробнее см. здесь https://habr.com/ru/post/99923/

```
font0.name = 'Times New Roman'
font0.colour_index = 2
font0.bold = True

style0 = xlwt.XFStyle()
style0.font = font0

style1 = xlwt.XFStyle()
style1.num_format_str = 'D-MMM-YY'

wb = xlwt.Workbook()
ws = wb.add_sheet('A Test Sheet')

ws.write(0, 0, 'Test', style0)
ws.write(1, 0, datetime.now(), style1)
ws.write(2, 0, 1)
ws.write(2, 1, 1)
ws.write(2, 2, xlwt.Formula("A3+B3"))

wb.save('example.xls')
```

SciPy — библиотека для языка программирования Python с открытым исходным кодом, предназначенная для выполнения научных и инженерных расчётов. Основные возможности – поиск минимумов и максимумов функций; вычисление интегралов функций; поддержка специальных и статистических функций и др. Команда подключения статистического пакета:

from scipy import stats.

Statsmodels.api – библиотека для языка программирования Python с открытым исходным кодом, предназначенная для проверки различных статистических гипотез. Команда подключения:

import statsmodels.api as sm.

4 Задания для выполнения работы

Разработать проект компьютерную реализацию системы обработки естественных языков, которая бы автоматизировала процесс определения тематики сообщества в социальной сети посредством реализации алгоритма, основанного на методе латентного распределения Дирихле, и оснащение ИС графическим интерфейсом.

1. Провести анализ предметной области.
2. Исследовать существующие способы построения информационно-поисковых программ в указаной предметной области.
3. Произвести моделирование системы обработки естественных языков.
4. Сформулировать техническое задание
5. Проектированиение (построение диаграмм)
6. Обосновать выбор программных инструментов. В качестве инструментальных средств выбрать программу PyCharm на языке программирования Python.

7. Реализовать алгоритм определения тематики сообществ, основанный на методе латентного распределения Дирихле а .

8. Произвести апробацию программнго обеспечения

Методика выполнения

1. **Анализ предметной области**.

Социальные сети в наше время являются уникальным источником данных об интересах людей, вследствие чего такая область, как анализ социальных сетей набирает стремительную популярность. Изучение социальных сетей как научное направление возникло на стыке ряд научных дисциплин – социология, дискретная математика, Computer Science (алгоритмы на графах и сетях). К сожалению, на данный момент четкого математического определения онлайн сообщества не существует. Обычно про сообщество говорят как про некоторую группу узлов или некоторую группу людей, которые связаны внутри группы сильнее между собой, чем со всем остальным миром. Однако, для анализа социальных данных требуются специализированные программные продукты, многие из которых создаются самими авторами перед проведением очередных исследований в данной области.

2. **Исследование существующих способов** построения информационно-поисковых программ в указаной предметной области.

Анализ социальных сетей широко используется в ряде приложений и дисциплин. Некоторые распространённые приложения сетевого анализа включают в себя сбор и накопление данных, моделирование распространения сети, моделирование сети и выборок, анализ характерных признаков и поведения пользователя, ресурсная поддержка, обеспечиваемая сообществом, анализ взаимодействия на основе местоположения, социальный обмен и отбор, развитие систем рекомендаций, а также прогнозирование связей и анализ объектов. В частном секторе фирмы используют анализ социальных сетей для поддержки такой деятельности, как взаимодействие и анализ клиентов, маркетинг и бизнес-аналитика. Использование анализа социальных сетей государственным сектором включает в себя развитие стратегий участия руководства, анализ индивидуального и группового участия, использование средств массовой информации и основанное на сообществах решение проблем.

Задача определения тематики сообщества в социальной сети сводится к тематическому моделированию коллекции документов, которые представляют собой тексты постов в сообществе.

Тематическое моделирование — это способ семантического анализа коллекции текстовых документов. Тематическая модель позволяет для каждого документа найти темы, которые его описывают, и кроме того показывает, какие слова характеризуют ту или иную тему. Другими словами, это более компактное представление большого набора текстов в виде нескольких тем. С математической точки зрения тематическая модель — это еще один вид матричного разложения, где в качестве исходной матрицы выступает матрица частот слов в документах.

Задана коллекция текстовых документов. Каждый документ представляет собой последовательность слов. Предполагается что каждый документ может относиться к одной или нескольким темам. Темы отличаются друг от друга различной частотой употребления слов. Для нахождения темы необходимо произвести действия, описанные на рисунке 6.8.

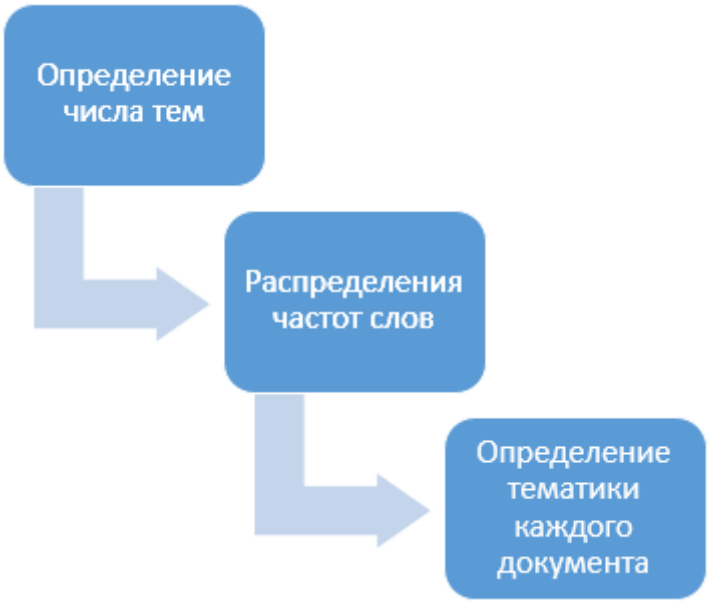

Рисунок 6. 8 – Этапы определения темы

3. **Моделирование** Данная задача может рассматриваться как задача одновременной кластеризации документов и слов по одному и тому же множеству кластеров, называемых темами.

Обычно строится мягкая, то есть документ может принадлежать нескольким темам в различной степени.

Целью построения тематической модели может быть, как непосредственно выявление множества латентных тем, так и решение различных дополнительных задач.

Примеры дополнительных задач:

- ранжировать документы по степени релевантности заданной теме (тематический поиск);
- ранжировать документы по степени тематического сходства с заданным документом или его фрагментом;
- построить иерархический тематический каталог коллекции документов и выработать правила каталогизации новых документов;
- определить, как темы изменялись со временем (предполагается, что для каждого документа известно время его создания);
- определить тематику авторов (предполагается, что для каждого документа известен список авторов);
- определить тематику различных сущностей (entities), связанных с документами (например, журналов, конференций, организаций, стран);

-разбить документ на тематически однородные фрагменты.

Типичные приложения:

- анализ коллекций научных статей;
- анализ новостных потоков;
- рубрикация коллекций изображений, видео, музыки;
- аннотация генома и другие задачи биоинформатики;
- коллаборативная фильтрация.

Базовые вероятностные тематические модели основаны на следующих предположениях.

- порядок документов в коллекции не важен;
- порядок слов в документе не важен, документ — «мешок слов» (bag of words);
- слова, встречающиеся в большинстве документов, не важны для определения тематики, их обычно исключают из словаря и называют стоп-словами;
- слово в разных формах — это одно и то же слово;
- коллекцию документов можно рассматривать как простую выборку пар «документ–слово» ;
- каждая тема описывается неизвестным распределением на множестве слов ;
- каждый документ описывается неизвестным распределением на множестве тем;
- гипотеза условной независимости: $p(w|t,d) = p(w|t)$.

Построить тематическую модель — значит, найти матрицы$F = \left|\left|p(w|t)\right|\right|$ и $O = ||p(t|d)||$ по коллекции D.

В более сложных вероятностных тематических моделях некоторые из этих предположений заменяются более реалистичными. Например, вместо модели «мешка слов» может использоваться марковская цепь; множество документов может рассматриваться как упорядоченное по времени их создания, и т. д.

Вероятностный латентный семантический анализ (probabilistic latent semantic analysis, PLSA) предложен Томасом Хофманном в 1999 году [4].

Вероятностная модель появления пары «документ–слово» может быть записана двумя эквивалентными способами:

$$p(d,w) = \sum_{t\epsilon T} p(t)p(w|t)p(d|t) = \sum_{t\epsilon T} p(d)p(w|t)p(t|d), \qquad (2.1)$$

где T — множество тем;

p(t) — неизвестное априорное распределение тем во всей коллекции;

p(d) — априорное распределение на множестве документов, эмпирическая оценка $(d) = n_d/n$, где n — суммарная длина всех документов;

p(w) — априорное распределение на множестве слов, эмпирическая оценка $p(w) = n_w/n$, где n_w — число вхождений слова w во все документы;

Искомые условные распределения выражаются по формуле Байеса:

$$p(w|t) = \frac{p(t|w)p(w)}{\sum_{w'} p(t|w')p(w')}; p(t|d) = \frac{p(d|t)p(t)}{\sum_{t'} p(d|t')p(t')}.$$

(2.2)

Основные недостатки PLSA:

Число параметров растёт линейно по числу документов в коллекции, что может приводить к переобучению модели.

При добавлении нового документа d в коллекцию распределение невозможно вычислить по тем же формулам, что и для остальных документов, не перестраивая всю модель заново.

Метод латентного размещения Дирихле (latent Dirichlet allocation, LDA) предложен Дэвидом Блеем в 2003 году. В этом методе устранены основные недостатки PLSA.

Метод LDA основан на той же вероятностной модели:

$$p(d, w) = \sum_{t \epsilon T} p(d)p(w|t)p(t|d), \quad (2.3)$$

при дополнительных предположениях:

- векторы документов порождаются одним и тем же вероятностным распределением на нормированных T-мерных векторах; это распределение удобно взять из параметрического семейства распределений Дирихле;

- векторы тем порождаются одним и тем же вероятностным распределением на нормированных векторах размерности W; это распределение удобно взять из параметрического семейства распределений Дирихле.

Для идентификации параметров модели LDA по коллекции документов применяется самплирование Гиббса, вариационный байесовский вывод или метод Expectation-Propagation.

Неотрицательное матричное разложение (НМР), а также неотрицательное приближение матрицы, это группа алгоритмов в мультивариантном анализе и линейной алгебре, в которых матрица V разлагается на (обычно) две матрицы W и H, со свойством, что все три матрицы имеют неотрицательные элементы. Эта неотрицательность делает получившиеся матрицы более простыми для исследования. В приложениях, таких как обработка спектрограмм аудиосигнала или данных мускульной активности, неотрицательность свойственна рассматриваемым данным. Поскольку задача в общем случае неразрешима, её обычно численно аппроксимируют.

НМР нашёл применение в таких областях как астрономия, компьютерное зрение, кластеризация документов и т.д.

Метод латентного распределения был выбран благодаря возможности быстрого извлечения темы из любого сообщества, без предварительного обучения модели машинного обучения, что многократно усложняет процесс и требует от пользователя более глубоких знаний программирования.

Также его преимуществом перед другими алгоритмами является то, что при добавлении нового документа d в коллекцию распределение возможно вычислить по тем же формулам, что и для остальных документов, не перестраивая всю модель заново.

При создании приложения по определению тематики необходимо использовать клиент-серверную архитектуру (Рисунок 6.9).

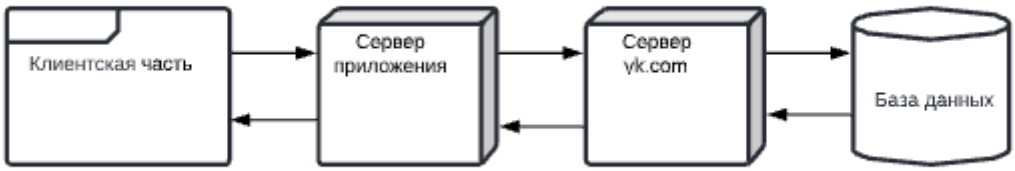

Рисунок 6.9 – Архитектура клиент-серверного приложения

Исходя из рисунка присутствует клиентская часть, которая получает от пользователя данные, в данном случае доменное имя сообщества, для которого производится тематическое моделирование. Затем она передает данные серверу приложения, который обращается к серверу социальной сети посредством API и после парсинга необходимой информации производит обработку данных с помощью алгоритма LDA. Чтобы задействовать методы тематического моделирования в приложении, нужно создать настраиваемый конвейер, который будет экстраполировать темы из неструктурированных текстовых данных. Конвейер для тематического моделирования изображен на рисунке 3.

В ходе предобработки происходит стэмминг. Стемминг — это процесс нахождения основы слова для заданного исходного слова. Основа слова не обязательно совпадает с морфологическим корнем слова.

Для улучшения выделения тематик, необходимо выделить коллокации. Коллокация - словосочетание, имеющее признаки синтаксически и семантически целостной единицы, в котором выбор одного из компонентов осуществляется по смыслу, а выбор второго зависит от выбора первого (например, ставить условия — выбор глагола ставить определяется традицией и зависит от существительного условия, при слове предложение будет другой глагол — вносить). Коллокация представляет собой обычную N-грамму.

N-грамма — последовательность из n элементов. С семантической точки зрения, это может быть последовательность звуков, слогов и т.д.

Рисунок 6.10 – Конвейер тематического моделирования

4. **Пример технического задания на разработку системы обработки естественных языков:**

Проектируемая информационная система предназначена для автоматизации процесса извлечения тематики сообщества в социальной сети. Система должна иметь следующий функционал:

- возможность ввода данных в поле графического интерфейса. В качестве идентификатора отдельного сообщества выступает доменное имя группы;

- возможность информативного вывода с визуализацией результата моделирования и возможностью увидеть все выделенные темы в виде кластеров, содержащих ключевые слова. По ним пользователь может в дальнейшем проводить расширенный анализ извлеченной информации.

Для вышеперечисленного необходимо реализовать клиент-серверное приложение с API, позволяющим извлекать данные из социальной сети и удобный графический интерфейс.

Пользователь должен иметь только возможность ввода данных и просмотра результата моделирования.

Интерфейс должен содержать только поле ввода данных (доменное имя группы) и выводить в браузер графики тематических моделей и диаграммы частот ключевых слов.

Для работы с приложением пользователю необходимо располагать интерпретатором Python на рабочем компьютере, а также браузером Internet Explorer начиная с версии 9.0.

5. Проектирование

Построение модели ПО, диаграммы классов, выбор готовых и описание собственных алгоритмов обработки данных были описаны с помощью нотации описания бизнес-процессов UML.

На рисунке 6.11 представлена диаграмма прецендентов для описания функциональных возможностей системы.

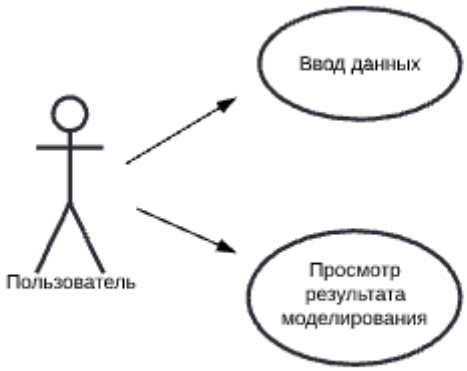

Рисунок 6.11 – Диаграмма прецедентов

В ИС присутствует пользователь, который имеет возможность ввести данные используя графический интерфейс и вывести результат моделирования в браузере.

На рисунке 5 представлена диаграмма последовательности для демонстрации последовательности передачи информации.

Как видно из рисунка процесс можно разделить на следующие этапы:

Пользователь вводит доменное имя сообщества, после чего данные обрабатываются на сервере.

Сервер приложения отправляет запрос через API серверу социальной сети (в данном случае сервер vk.com) и получает необходимые данные.

Серверный скрипт с реализованным алгоритмом обрабатывает данные и выводит графическую модель в браузере.

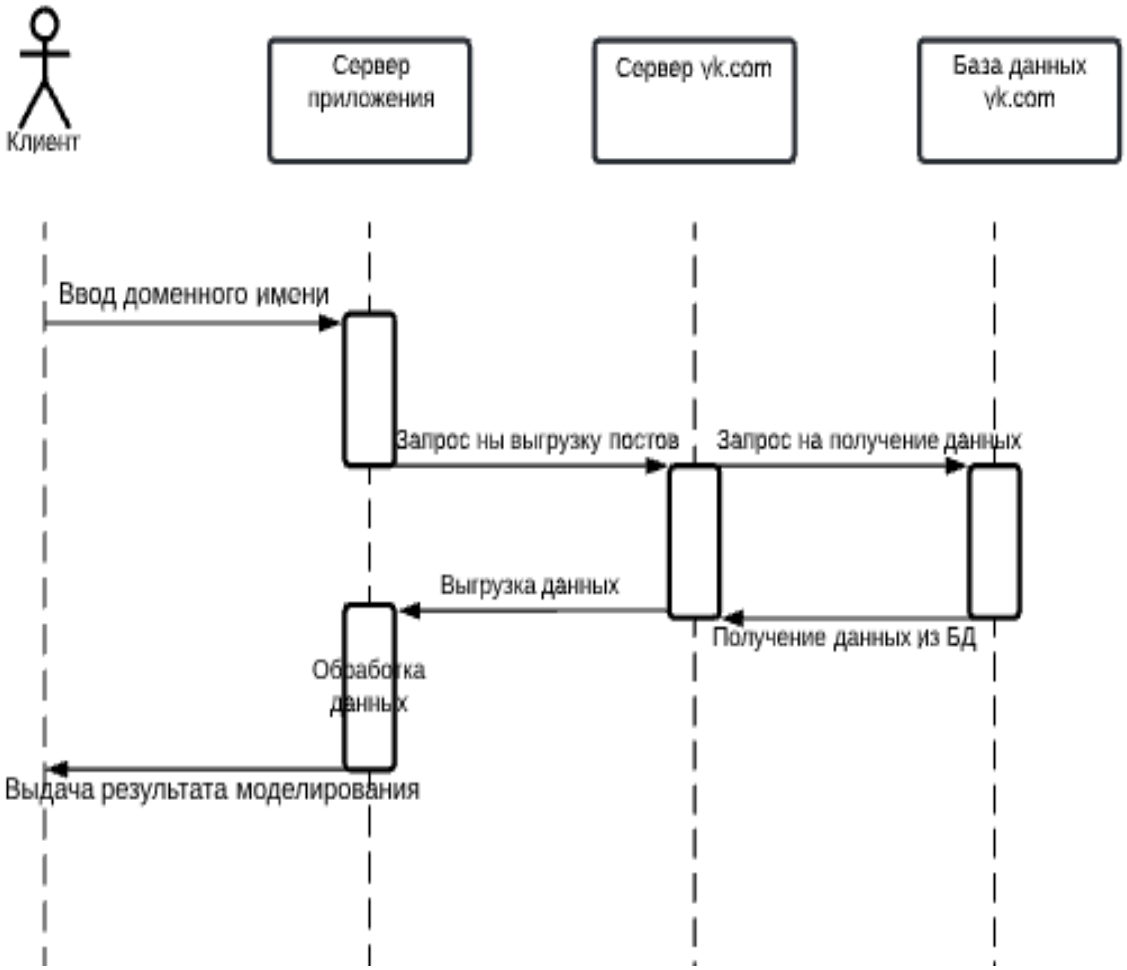

Рисунок 6.12– Диаграмма последовательностей

В разрабатываемой ИС можно выделить несколько состояний, показанных на рисунке 6.12.

Рисунок 6.13 – Диаграмма состояний

Выбор программных инструментов

В настоящее время существует широкий список языков программирования, которые подходят для разработки серверного приложения. В таблице 4 приведены несколько таких языков.

Таблица 6.1 – Сравнительный анализ ЯП для написания серверной части

ЯП	Год выпуска	Enterprise	Наличие готовых инструментов	Рейтинг
C#	2000	Да	Есть	3
Java	1995	Да	Есть	2
Python	1991	Нет	Есть	4
JavaScript	1995	Нет	Есть	1
TypeScript	2012	Нет	Есть	5

В качестве языка разработки клиентской и серверной частей был выбран язык Python благодаря следующим преимуществам:

Гибкость — позволяет программировать различные архитектуры серверных приложений, использовать различные парадигмы программирования.

Расширяемость — существуют библиотеки и фреймворки под любой тип задач и надобностей. Также огромным плюсом является то, что можно использовать C код из Python.

Простота синтаксиса. Из синтаксиса было убрано все лишнее, код чист и понятен без лишних скобок и выражений [7].

Интерпретируемость. Интерпретатор Python существует для всех популярных платформ и по умолчанию входит в большинство дистрибутивов Linux, а значит есть на большинстве серверов «из коробки».

PEP — единый стандарт для написания кода, что делает код поддерживаемым и читабельным даже при переходе от одного программиста к другому. Это поддерживает популярность Python [8].

OpenSource — код интерпретатора Python является открытым, что позволяет любому, кто заинтересован в развитии языка поучаствовать в его разработке и улучшить его. Если смотреть детали релиза одной из версий языка, то можно заметить, что огромные части нового функционала реализованы сторонними разработчиками.

Но главным образом благодаря наличию библиотек, упрощающих и ускоряющих обработку естественного языка:

- gensim - содержит все алгоритмы тематического моделирования;
- pandas - необходима для работы с корпусом языка;
- nltk - содержит алгоритмы лемматизаций и словарь стоп слов;
- pyLDAvis - плагин визуализации модели LDA;
- matplotlib - библиотека визуализации.

7. Компьютерная реализация приложения

1. Импортировать необходимые библиотеки и пакеты с помощью команды import.

```
import requests
import csv
import time
import pandas as pd
import tkinter as tk
import nltk
from nltk.corpus import stopwords
import re, string
from nltk.stem.snowball import SnowballStemmer
from gensim.models import Phrases
from gensim.corpora.dictionary import Dictionary
from numpy import array
import threading
import os
```

```
from gensim.models.ldamulticore import LdaMulticore
#import pyLDAvis.gensim_models as gensim
import pyLDAvis.gensim as gensim
import pyLDAvis
import webbrowser
```

2. Затем инициализировать интерфейс с помощью функций def get_clear_domain() и def handle_click_start().

```
def get_clear_domain():
    return entry.get().replace('vk.com/', '').replace('https://', '')

def handle_click_start():
    try:
        parse_btn['state'] = 'disabled'
        parse_btn['text'] = 'Ожидайте...'
        entry['state'] = 'disabled'
        result_label['text'] = ''
        global regex, russian_stopwords, stemmer

        all_posts = take_posts()
        file_writer(all_posts)

        df = pd.read_csv(r'%s.csv' % get_clear_domain(), encoding='utf-8',
sep=',',usecols=[1])
        nltk.download("stopwords")

        russian_stopwords = stopwords.words("russian")# собираем стоп слова
        regex = re.compile('[%s]' % re.escape(string.punctuation)) # компилим
regexp выражение
        stemmer = SnowballStemmer("russian") # инициализируем стэмминг


        df['text'] = df['text'].apply(lambda x: preprocessing(x))

        text_clean= []
        for index, row in df.iterrows():
            text_clean.append(row['text'].split())


        bigram = Phrases(text_clean) # Создаем биграммы на основе корпуса
        trigram = Phrases(bigram[text_clean])# Создаем триграммы на основе
корпуса
```

```
        for idx in range(len(text_clean)):
            for token in bigram[text_clean[idx]]:
                if '_' in token:
                    # Токен это би грамма, добавим в документ.
                    text_clean[idx].append(token)
            for token in trigram[text_clean[idx]]:
                if '_' in token:
                    # Токен это три грамма, добавим в документ.
                    text_clean[idx].append(token)

        dictionary = Dictionary(text_clean)

        #Создадим словарь и корпус для lda модели
        corpus = [dictionary.doc2bow(doc) for doc in text_clean]
        print('Количество уникальных токенов: %d' % len(dictionary))
        print('Количество документов: %d' % len(corpus))
        #from gensim.models.ldamulticore import LdaMulticore
        model=LdaMulticore(corpus=corpus,id2word=dictionary, num_topics=2)
        model.show_topics()

        data = gensim.prepare(model, corpus, dictionary)
        pyLDAvis.save_html(data, r'%s.html' % get_clear_domain())
        url = 'file:///' + os.path.realpath(r'%s.html' % get_clear_domain())
        webbrowser.open(url, new=2)  # open in new tab

        print(data)
        result_label['text'] = 'Парсинг %s прошел успешно!' %
get_clear_domain()
        parse_btn['state'] = 'active'
        parse_btn['text'] = 'Начать парсинг'
        entry['state'] = 'normal'
        entry.delete(0, 'end')

    except Exception as err:
        print(err)
        result_label['text'] = 'Парсинг %s прошел с ошибкой:%s' %
(get_clear_domain(), str(err))
        parse_btn['state'] = 'active'
        parse_btn['text'] = 'Начать парсинг'
        entry['state'] = 'normal'
        entry.delete(0, 'end')
```

3. С помощью команды nltk.download загрузить стоп-слова из библиотеки nltk.

```
nltk.download("stopwords")
```

4. В функции def_take_posts() реализовать парсер, который производит извлечение текстов записей в социальной сети и выводит в виде csv файла.

```
def take_posts():
    token =
'254b23ce254b23ce254b23ce08253dd0ec2254b254b23ce45189f38c041c0cb6
bfc0dac'
    version = 5.103
    domain = get_clear_domain()
    count = 100
    offset = 0
    all_posts = []

    while offset < 1000:
        print(offset)

print('https://api.vk.com/method/wall.get?access_token=%s&v=%s&domain=
%s&count=%s' % (token, version, domain, count))
        response = requests.get('https://api.vk.com/method/wall.get',
                                params={'access_token': token, 'v': version, 'domain':
domain, 'count': count,
                                        'offset': offset})
        print(response.status_code)
        if 'error' in  response.json() and 'error_msg' in response.json()['error']:
            raise Exception(response.json()['error']['error_msg'])

        data = response.json()['response']['items']
        offset += 100
        all_posts.extend(data)
        time.sleep(0.5)
    return all_posts
```

5. Затем считанные данные подать на вход алгоритма. В функции def_preprocessing разработать процедуру препроцессинга данных и стемматизации.

```
def preprocessing(text):
    import sys
```

```
    if not text:
        return " "
    non_bmp_map = dict.fromkeys(range(0x10000, sys.maxunicode + 1),
0xfffd)
    text = str(text).translate(non_bmp_map)

    text = regex.sub('', text) # удаляем пунктуацию
    text = [token for token in text.split() if token not in russian_stopwords] #
Удаляем стоп слова
    text = [stemmer.stem(token) for token in text] # Выполняем стэмминг
    text = [token for token in text if token] # Удаляем пустые токены
    return ' '.join(text)
```

6. С помощью процедур text_clean, dictionary и model разработать дальнейшую обработку текста, выделение n-грамм и построение модели.

7. С помощью метода pyLDAvis.enable_notebook() построить визуализацию работы алгоритма.

```
    pyLDAvis.save_html(data, r'%s.html' % get_clear_domain())
    url = 'file:///' + os.path.realpath(r'%s.html' % get_clear_domain())
    webbrowser.open(url, new=2)
```

Код на Python

```
import requests
import csv
import time
import pandas as pd
import tkinter as tk
import nltk
from nltk.corpus import stopwords
import re, string
from nltk.stem.snowball import SnowballStemmer
from gensim.models import Phrases
from gensim.corpora.dictionary import Dictionary
from numpy import array
import threading
import os
from gensim.models.ldamulticore import LdaMulticore
#import pyLDAvis.gensim_models as gensim
import pyLDAvis.gensim as gensim
import pyLDAvis
import webbrowser

class Worker(threading.Thread):
    def run(self):
        handle_click_start()
```

```
def run_in_back():
    w = Worker()
    w.start()

regex = ''
russian_stopwords = ''
stemmer = ''
parse_btn = ''
entry = ''
result_label = ''

def get_clear_domain():
    return entry.get().replace('vk.com/', '').replace('https://', '')

def handle_click_start():
    try:
        parse_btn['state'] = 'disabled'
        parse_btn['text'] = 'Ожидайте...'
        entry['state'] = 'disabled'
        result_label['text'] = ''
        global regex, russian_stopwords, stemmer

        all_posts = take_posts()
        file_writer(all_posts)

        df = pd.read_csv(r'%s.csv' % get_clear_domain(), encoding='utf-8', sep=',',usecols=[1])
        nltk.download("stopwords")

        russian_stopwords = stopwords.words("russian")# собираем стоп слова
        regex = re.compile('[%s]' % re.escape(string.punctuation)) # компилим regexp выражение
        stemmer = SnowballStemmer("russian") # инициализируем стэмминг


        df['text'] = df['text'].apply(lambda x: preprocessing(x))

        text_clean= []
        for index, row in df.iterrows():
            text_clean.append(row['text'].split())


        bigram = Phrases(text_clean) # Создаем биграммы на основе корпуса
        trigram = Phrases(bigram[text_clean])# Создаем триграммы на основе корпуса

        for idx in range(len(text_clean)):
            for token in bigram[text_clean[idx]]:
                if '_' in token:
                    # Токен это би грамма, добавим в документ.
                    text_clean[idx].append(token)
            for token in trigram[text_clean[idx]]:
                if '_' in token:
                    # Токен это три грамма, добавим в документ.
```

```
                text_clean[idx].append(token)

        dictionary = Dictionary(text_clean)

        #Создадим словарь и корпус для lda модели
        corpus = [dictionary.doc2bow(doc) for doc in text_clean]
        print('Количество уникальных токенов: %d' % len(dictionary))
        print('Количество документов: %d' % len(corpus))
        #from gensim.models.ldamulticore import LdaMulticore
        model=LdaMulticore(corpus=corpus,id2word=dictionary, num_topics=2)
        model.show_topics()

        data = gensim.prepare(model, corpus, dictionary)
        pyLDAvis.save_html(data, r'%s.html' % get_clear_domain())
        url = 'file:///' + os.path.realpath(r'%s.html' % get_clear_domain())
        webbrowser.open(url, new=2)  # open in new tab

        print(data)
        result_label['text'] = 'Парсинг %s прошел успешно!' % get_clear_domain()
        parse_btn['state'] = 'active'
        parse_btn['text'] = 'Начать парсинг'
        entry['state'] = 'normal'
        entry.delete(0, 'end')

    except Exception as err:
        print(err)
        result_label['text'] = 'Парсинг %s прошел с ошибкой:%s' % (get_clear_domain(), str(err))
        parse_btn['state'] = 'active'
        parse_btn['text'] = 'Начать парсинг'
        entry['state'] = 'normal'
        entry.delete(0, 'end')

def take_posts():
    token = '254b23ce254b23ce254b23ce08253dd0ec2254b254b23ce45189f38c041c0cb6bfc0dac'
    version = 5.103
    domain = get_clear_domain()
    count = 100
    offset = 0
    all_posts = []

    while offset < 1000:
        print(offset)
        print('https://api.vk.com/method/wall.get?access_token=%s&v=%s&domain=%s&count=%s'
% (token, version, domain, count))
        response = requests.get('https://api.vk.com/method/wall.get',
                                params={'access_token': token, 'v': version, 'domain': domain, 'count': count,
                                        'offset': offset})
        print(response.status_code)
        if 'error' in  response.json() and 'error_msg' in response.json()['error']:
            raise Exception(response.json()['error']['error_msg'])

        data = response.json()['response']['items']
```

```
        offset += 100
        all_posts.extend(data)
        time.sleep(0.5)
    return all_posts


def file_writer(data):
    with open(r'%s.csv' % get_clear_domain(), 'w', encoding="utf-8") as file:
        a_pen = csv.writer(file)
        a_pen.writerow(('likes', 'text'))
        for post in data:
            a_pen.writerow((post['likes']['count'], post['text']))

def preprocessing(text):
    import sys
    if not text:
        return " "
    non_bmp_map = dict.fromkeys(range(0x10000, sys.maxunicode + 1), 0xfffd)
    text = str(text).translate(non_bmp_map)

    text = regex.sub('', text) # удаляем пунктуацию
    text = [token for token in text.split() if token not in russian_stopwords] # Удаляем стоп слова
    text = [stemmer.stem(token) for token in text] # Выполняем стэмминг
    text = [token for token in text if token] # Удаляем пустые токены
    return ' '.join(text)


if __name__ == '__main__':
    window = tk.Tk()
    window.geometry("400x200")

    w = 400
    h = 400

    ws = window.winfo_screenwidth()
    hs = window.winfo_screenheight()
    x = (ws/2) - (w/2)
    y = (hs/2) - (h/2)

    window.geometry('%dx%d+%d+%d' % (w, h, x, y))



    label = tk.Label(text="url группы")

    entry = tk.Entry()

    parse_btn = tk.Button(
          text="Начать парсинг",
          command=run_in_back

    )
```

```
label.pack()
entry.pack()
parse_btn.pack()
result_label = tk.Label(text="")
result_label.pack()


window.mainloop()
```

8. Апробация

Пример описания работы приложения:

Для начала работы с приложением необходимо иметь на ПК интерпретатор Python не ниже третьей версии. Запуск проекта можно произвести как с помощью IDE, так и через консоль.

При запуске приложения выводится графический интерфейс (Рисунок 6.13), в поле ввода которого пользователь может ввести доменное имя анализируемого сообщества.

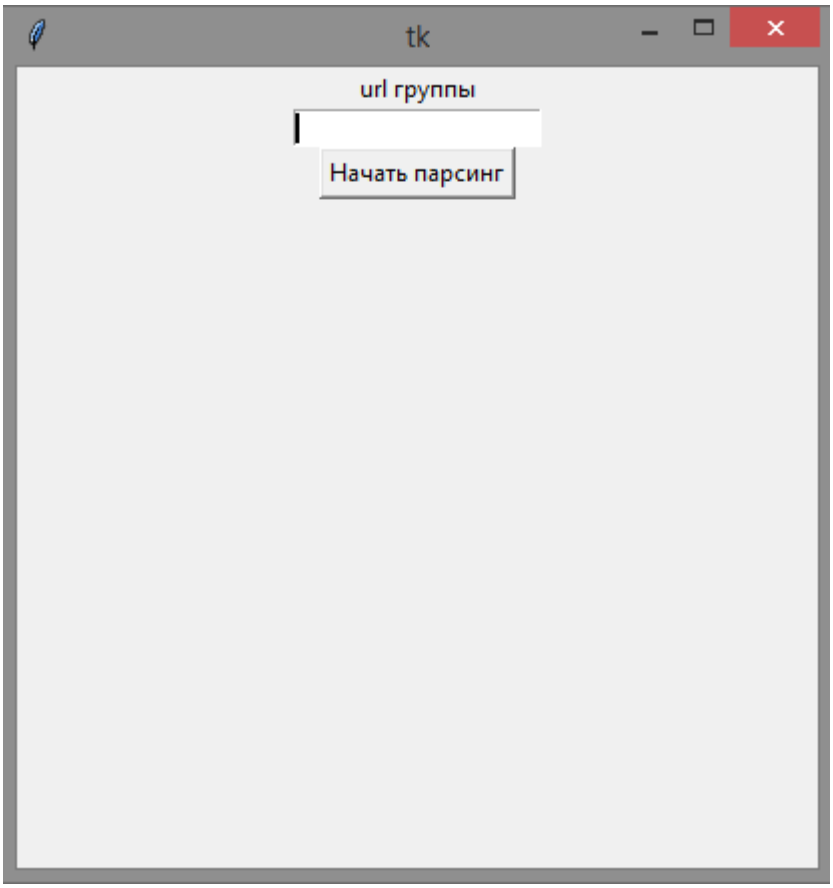

Рисунок 6.3 – Начальный интерфейс

При вводе некорректных данных (Рисунок 6.14) выводится текст ошибки.

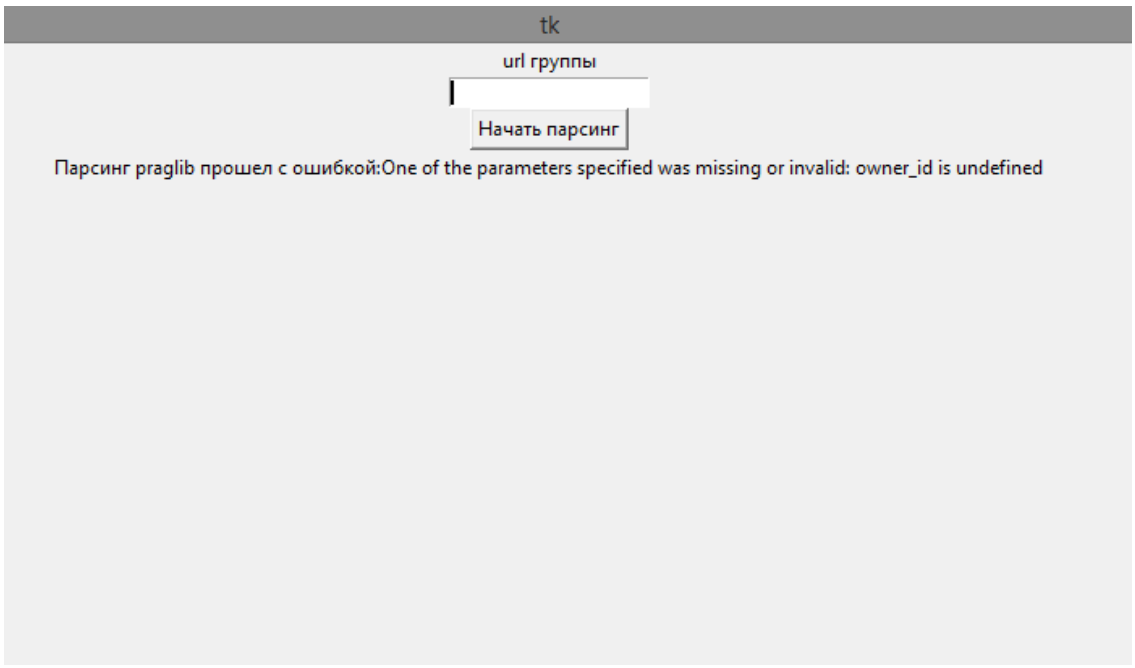

Рисунок 6.14 – Ввод некорректных данных

При корректном вводе (Рисунок 6.15) начнется процесс парсинга данных, который займет какое-то время.

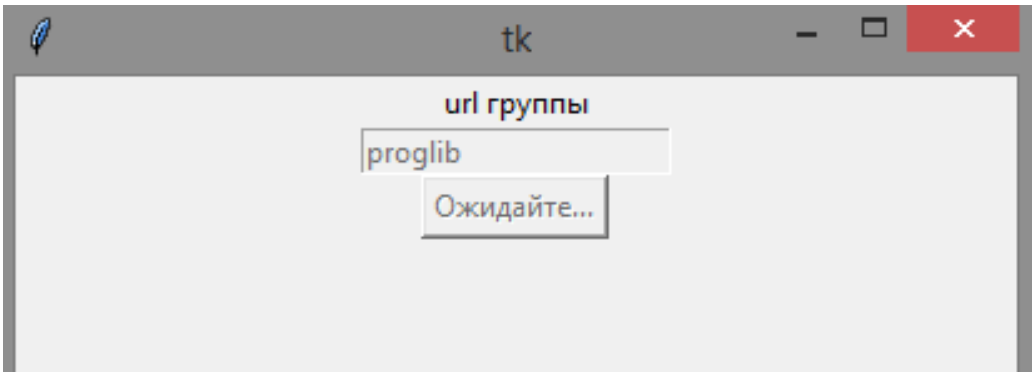

Рисунок 6.15 – Запуск парсинга

После окончания процесса моделирования выводится интерактивная графическая модель в браузере с выделенными в виде кластеров темами, а также наиболее часто встречающимися словами внутри темы (Рисунок 6.16).

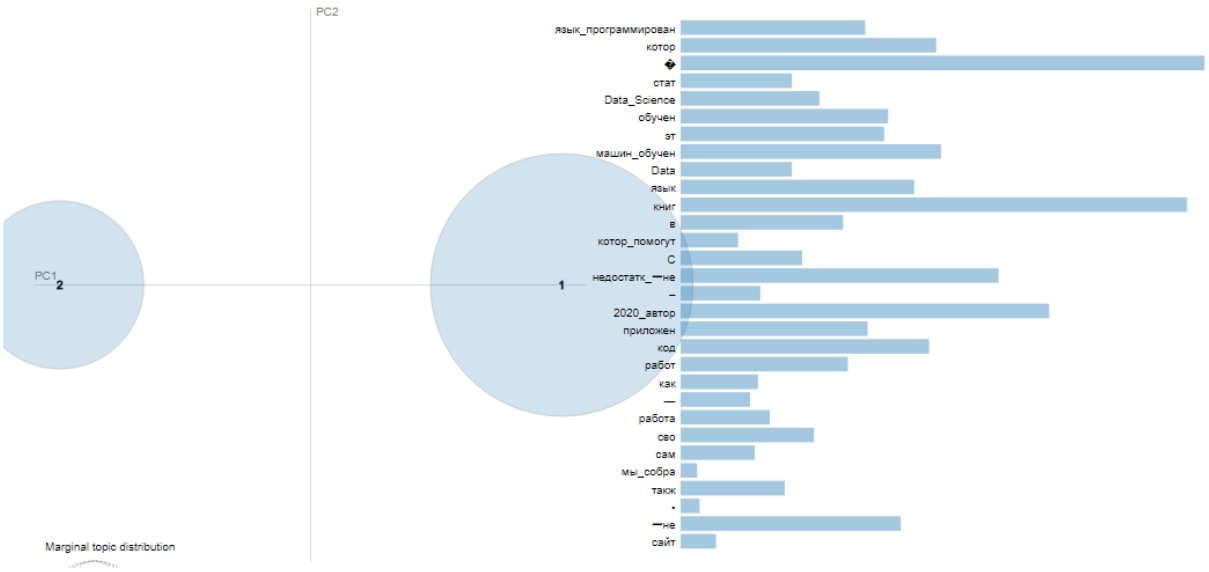

Рисунок 16.16 – Вывод графической модели в браузер

При наведении курсора на кластер можно получить список слов, принадлежащих именно ему (Рисунок 6.17).

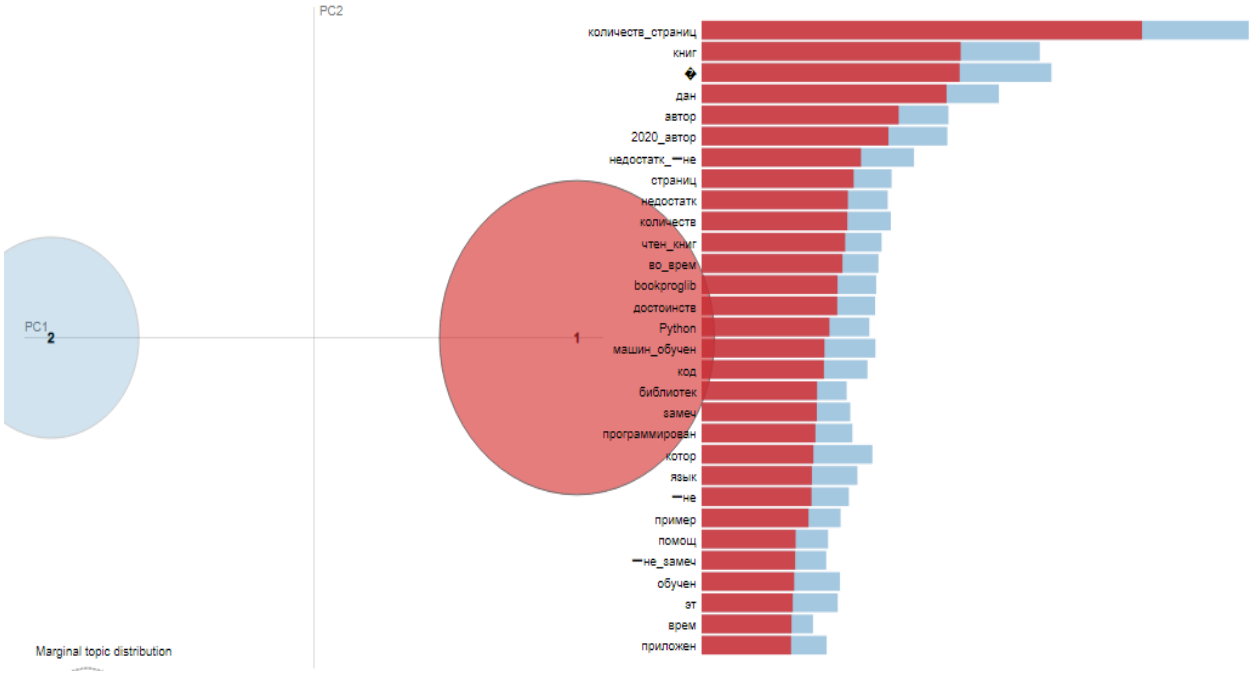

Рисунок 6.17 – Выбор кластера

На основании этой информации можно сделать вывод о принадлежности сообщества определенной тематике.

6 Контрольные вопросы

1) Расскажите об основные области применения обработки етественного языка на ЭВМ.
2) Каковы основные требования к процессу понимания запросов на уровне интерфейса на естественном языке в интеллектуальных системах?
3) Отобразите и поясните общую схему анализа высказывания.
4) Перечислите и дайте краткое описание состава моделей в текущей версии NLP Architect
5) Какие технологии и методы NLP вам известны?
6) Поясните лексическое значение слова и его описание средствами лингвистических информационных ресурсов.
7) Поясните фреймовое представление ситуации действительности и модель управления предикатного слова: сравнительный анализ.
8) В чем суть этапа синтаксического анализа входного предложения?
9) Каковы основные принципы построения правил и стратегий синтаксического анализа фраз естественного языка для задач компьютерной обработки текстов?
10) Каковы типы синтаксических фильтров?
11) Отобразите и поясните общую структуру алгоритма синтаксического анализа фразы русского языка (без рассмотрения оборотов).
12) Каким образом производится распознавание семантической эквивалентности и ситуация языкового употребления?
13) В чем суть е синонимических замен на уровне абстрактной лексики?
14) Расскажите об этапе семантического анализа входного предложения.
15) Каковы особенности интерфейса на естественном языке для интеллектуальной системы с фреймовой моделью в основе представления предметных знаний.?
16) Перечислите типы вопросительных ситуаций.
17) Как производится обработка пустых и функциональных предикатов на этапе семантического анализа входного предложения?
18) Как происходит построение семантического графа входного предложения. Замена обстоятельственных отношений семантическими отношениями при обработке предикатных слов в запросах к фреймовой сети?
19) Как происходит интерпретация входного предложения и синтеза семантического графа ответа?
20) Как производится синтез синтаксической структуры ответа?
21) Поясните механизм определение порядка слов и морфологический синтез словоформ ответа.