Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное учреждение высшего образования
«Кубанский государственный технологический университет»

Н.П. Орлянская, М.В. Янаева

ИНФОРМАЦИОННЫЙ ПОИСК И ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА

Конспект лекций
для студентов всех форм обучения по направлению
09.04.03 Прикладная информатика

Краснодар
2021

УДК 681.4
ББК 32.813
О-66

Орлянская Н.П., Янаева М.В. Информационный поиск и обработка естественного языка: конспект лекций / Кубанский государственный технологический университет – Краснодар: изд. ФГБОУ ВО «КубГТУ», 2021. - 187 с.

В конспекте лекций рассматриваются базовые вопросы информационного поиска и обработки естественного языка.

Предназначен для студентов высших учебных заведений, обучающихся по направлению подготовки 09.04.03 Прикладная информатика, уровень подготовки – магистратура всех форм обучения.

Ил. 47. Библиогр.: 46 назв.

Печатается по решению методического совета Кубанского государственного технологического университета.

Рецензенты:
д-р техн. наук, проф., проф. кафедры ИСП КубГТУ В.Н. Марков
канд. физико-математических наук, д-р экон. наук, проф., зав. кафедрой информационных систем КубГАУ Е.В. Попова

ISBN

Содержание

Предисловие

В первой конспекта лекций рассматриваются основные понятия и задачи информационного поиска, а также его практическая значимость. Рассматривается теоретическая база необходимая для построения полнофункциональных поисковых систем.

Вторая глава посвящена обработке естественного языка, распознаванию имен собственных и построению вопросно-ответных систем.

Каждая из лекций содержит план, основную часть, контрольные вопросы.

1 Информационный поиск

1.1 Лекция №1 Введение в информационный поиск и обработку естественного языка

План лекции

1. Основные понятия информационного поиска и обработки естественного языка.
2. Модели информационного поиска.
3. Булевская модель.

1.1.1 Основные понятия информационного поиска и обработки естественного языка

Стремительный рост Интернета и успешное развитие информационно-поисковых систем привели к тому, что современный информационный поиск как дисциплина включает широкий круг вопросов, связанных со сбором, хранением, поиском и представлением самой разнообразной информации; сюда же естественным образом относятся многие задачи автоматической обработки текста.

Поскольку объектом обработки выступают тексты естественного языка, развитие данного направления невозможно без базовых знаний в области общей лингвистики (языкознания). Лингвистика изучает общие законы естественного языка — его структуру и функционирование, и включает такие области:

— фоне́тика — раздел лингвистики, изучающий звуки речи и звуковое строение языка;

— морфология (лингвистика) — раздел грамматики, изучающий части речи, их категории и формы слов. Например: Морфология русского языка — дисциплина, основным объектом изучения которой являются слова русского языка и их значимые части;

— синтаксис — это совокупность правил, теоретических систем и языковых процессов, упорядочивающих и изучающих структуру предложений в каком-либо языке;

— сема́нтика (от др.-греч. σημαντικός «обозначающий») — раздел лингвистики, изучающий смысловое значение единиц языка;

— прагматика — раздел языкознания, изучающий условия использования говорящими языковых знаков, воздействие прагматики определяется содержанием и оформлением высказывания.

Информационный поиск – выбор документов из большой коллекции, удовлетворяющих потребности пользователя, сформулированной в виде короткого запроса на естественном языке.

Запрос –способ выражения информационных потребностей пользователя поисковой системы, представленный формально [80]. Для его формулировки этого специальный язык поисковых запросов, составленный по определенным правилам.

Объект запроса — это информационная сущность, которая хранится в базе автоматизированной системы поиска. Несмотря на то, что наиболее распространенным объектом запроса является текстовый документ, не существует никаких принципиальных ограничений.

Индексирование:

— поиск по большим коллекциям не может осуществляться в режиме реального времени;

— для быстрого поиска коллекция предварительно обрабатывается и по ней строится индекс(ы) – набор атрибутов, которые упорядочены в удобном для поиска порядке;

— в случае полнотекстового поиска такими атрибутами являются слова (словосочетания), приведенные к нормальной форме.

Нечёткий поиск:

— процедура поиска по запросам пользователей, содержащим опечатки и неточным запросам, в которых пользователь может скрыть часть слова за специальным знаком "*".

Релева́нтность (англ. *relevant*)

— степень соответствия запроса и найденного, то есть уместность результата.

Ранжирование:

— отобор из них лучших результатов поиска и представление их пользователю в порядке "полезности".

Взвешивание:

— В индексе хочется учитывать не только сам факт вхождения слова в документ, но и «вес», т.е. информацию о частоте данного слова в документе.

— Однако саму по себе частоту использовать плохо, поскольку слова распределены в языке неравномерно: некоторые встречаются гораздо чаще других

Процесс индексирования можно разбить на пять шагов:

1) анализ структуры – выделение заголовков, абзацев и т.п.; удаление html-разметки и т.д;

2) токенизация – разбиение текста на слова, удаление знаков препинания;

3) удаление стоп-слов - высокочастотных служебных слов (предлогов, союзов и т.п.);

4) лемматизация – приведение слов к нормальной (например, словарной) форме;

5) взвешивание – частота слов в документе.

Структура индекса представлена на рисунке 1.1.

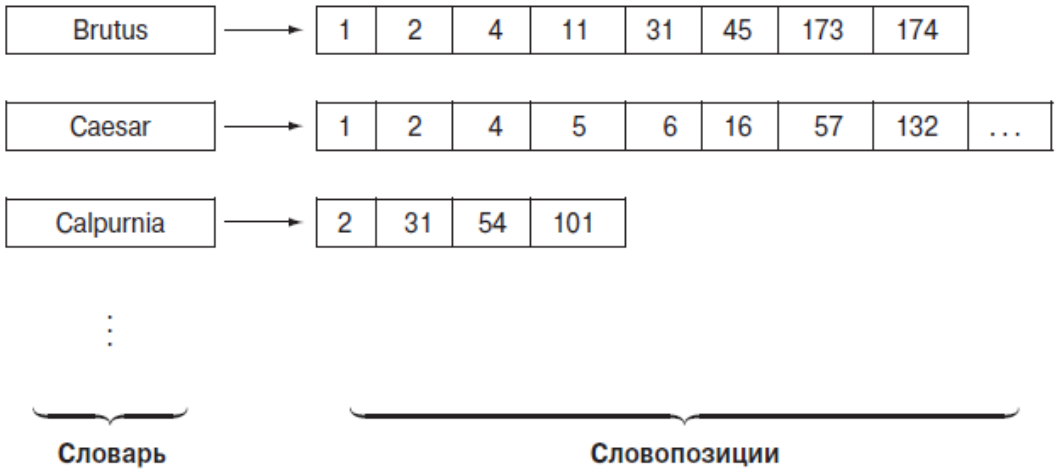


Рисунок 1.1 – Структура индекса

1.1.2 Модель информационного поиска

В состав модели информационного поиска входят форматы представления документов и запросов и функция соответствия документа запросу.

На рисунке 1.2 представлена схема основных классов моделей информационного поиска.

Рисунок 1.2 – Основные классы моделей информационного поиска

Основные задачи информационного поиска:

— классическая предназначена для поиска документов, удовлетворяющих запросу, в некоторой статической коллекции документов, например, в справочной коллекции операционной системы Windows.

— кластеризация документов –выявление групп семантически схожих документов среди определенного множества;

— классификация документов – определение тематики документа;

— фильтрация документов – разбиение множества документов на категории, т. е. автоматический выбор документов, соответствующих заданной тематике и отсев прочих документов.

1.1.3 Булевская модель

Одним из видов класса теоретико - множественных моделей является *булевская*. Она построена на основании теоретико- множественных операций: пересечение, объединение и вычитание.

В основе работы любой современной информационной поисковой системы лежат достаточно простые принципы, которые мы с вами сейчас и рассмотрим.

Начнем с того, что рассмотрим такой вырожденный случай, когда пользователь сформировал запрос, состоящий всего из одного слова. Можно достаточно просто обработать такой запрос. Достаточно взять весь корпус документов, который у есть, и начать перебирать документы по очереди. Каждый документ будем проверять, содержит ли он слово из запроса, и если содержит, то этот документ попадает в ответ поисковой системы.

Однако, что делать, если запрос пользователя сложней, если он состоит из нескольких терминов?

В таком случае на помощь придут булевы операции. В основном в информационном поиске используется три основные булевы операции:

1) логическое «и», случай, когда документ подходит нам, если он содержит и один и другой термин из запроса;

2) логическое «или». Случай, когда документ подходит нам, если он содержит хотя бы один термин из запроса либо оба термина;

3) отрицание. Это тот случай, когда мы должны выбрать все документы, которые не содержат слово, связанное с этим операндом.

Рассмотрим на конкретном примере применение данных операций.

Пользователь составил простой запрос из одного слова, но далее решил его расширить.

В итоге получаем следующий запрос: «найти такие документы, которые во-первых, содержат слово «лук», во-вторых, содержат одно из слов — «стрельба» либо «стрельбище», и абсолютно не содержат слово «растение»».

Можно сделать вывод, что пользователь хочет найти какую-то информацию, которая связана со стрельбой из лука.

Начнем обрабатывать запрос пользователя, постепенно увеличивая его. В самом начале обработаем только термин «лук» и найдем все документы, в которых присутствует это слово.

Дальше расширим этот запрос с помощью комбинации с логическим «или». Для того чтобы обработать ее, мы должны найти все документы, которые содержат одно слово и другое, а затем объединить их.

Так как в расширенном запросе присутствует логическое «и», необходимо пересечь это новоприобретенное объединение с тем множеством документов, в которых содержится слово «лук».

И уже итоговое пересечение и будет как раз ответом нашей поисковой системы на этот расширенный запрос. Но как мы помним, это не весь запрос. У нас есть еще третья часть с отрицанием. Как мы можем обработать отрицание? Необходимо найти все документы, в которых не содержится это слово, и уже полученное нами множество мы пересекаем с множеством, полученном на прошлом этапе, и в конечном счете получаем ответ на такой комплексный сложный запрос пользователя.

Очевидно, что каждый раз использовать весь корпус документа для того, чтобы обработать запрос пользователя, — это очень долго и накладно, поэтому давайте перейдем с вами к использованию такой сущности, как матрица инцидентности. Это особый вид матрицы, который содержит в себе информацию о том, какие слова в каких документах хранятся. Столбцы в этой матрице — это все документы нашего корпуса, строки — это все слова из этих документов (рис. 1.3).

На пересечении строки и столбца ставим либо единицу, либо ноль. Единицу мы ставим в том случае, если слово из этой строки встречается в документе из этого столбца.

Ноль ставим в противоположном случае: то есть, если слова из этой строки в данном документе нет. Как же в данном случае мы будем обрабатывать тот комплексный запрос, который мы рассматривали с вами в качестве примера в прошлый раз?

	doc_1	doc_2	doc_3	doc_4	doc_5	…
…	…	…	…	…	…	…
лук	1	1	0	0	1	…
стрельба	1	1	1	1	1	…
стрельбище	1	0	0	0	0	…
растение	1	0	0	0	0	…
…	…	…	…	…	…	…

Рисунок 1.3 – Матрица инцидентности

В данном случае будем работать с числовыми векторами. К ним применимы булевские операции и возможно получить какой-то результат. Результатом будет вектор, состоящий из нулей и единиц.

Теперь мы должны преобразовать его к виду, который был бы понятен пользователю. Для этого мы опять обращаемся к нашей матрице инцидентности: выбираем из нее те документы, которые хранятся в столбцах, чьи порядковые номера соответствуют единицам в итоговом векторе — так мы в итоге получаем множество документов, которые и являются ответом, который мы можем показать пользователю.

Очевидно, что в общем случае матрица инцидентности также не самый выгодный способ хранения информации, так как это достаточно разряженная структура: там очень много нулей, которые практически не несут для нас никакой полезной информации. Таким образом, мы можем перейти к использованию другой структуры, которая лучше отвечает нашим запросам, — это обратный индекс (рис.1.4). Это структура, которая хранит в себе весь словарь: все слова, которые есть в нашем корпусе документов и все документы, в которых эти слова встречаются.

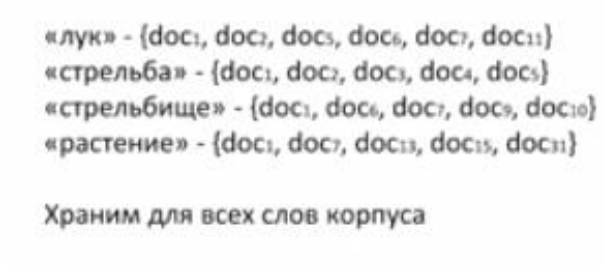


Рисунок 1.4 – Обратный индекс

Как видно, структура очень похожа на то множество документов, которые мы получали, пробуя свой первый наивный подход к обработке сложного запроса пользователя. Мы должны хранить такую информацию для всех слов нашего корпуса. Кроме того, можно сказать, что, кроме непосредственно самих слов и списка документов, в которых они встречаются, можно хранить также дополнительную информацию, которую мы впоследствии можем использовать, например, для чернового ранжирования: это может быть как позиция слова в документе, это может быть частота слова во всем корпусе, так и какая-то общая абстрактная популярность документа.

Недостаток булевской модели состоит в непригодности результатов поиска для ранжирования.

1.1.4 Лексикон и список слово позиций

Можно выделить следующие этапы построения инвертированного индекса:

— выбрать документы, подлежащие индексированию;

— токенизация;

— лингвистическая обработка лексем;

— индексация документов по каждому термину.

Разберем некоторые из этапов.

Токенизация (tokenization) — это процесс выделение лексем. Формально, лексема – это абстрактная единица морфологического анализа в лингвистике, примерно соответствующая совокупности форм одного слова.

Результатом лингвистической обработки являются классы эквивалентных лексем, инициирующие совокупности терминов. Это входная информация для индексирования и формирования инвертированных списков. Структура инвертированного списка, повышает скорость обработки запросов.

В начале цифровая информация в виде байтов должна быть преобразована в линейную последовательность символов. Эта операция напрямую связана с кодированием. Выявление кодировки можно осуществить основе машинного обучения, или эвристическими методами. Кодировка дает представление о языковом представлении документа.

В XML - документах ряд символов, например: «& атр;» декодируют специально. Кроме того, текстовую часть документа может зачастую отделяют от другого материала, который не подвергается обработке.

Предположим, что наши документы имеют вид списков символов. Не всегда документы представимы линейной последовательностью символов, например в арабских языках текст является двумерным и не имеет строгого порядка следования символов, как показано н а рисунке 1.5.

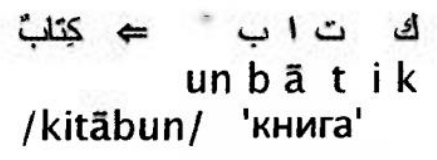


Рисунок 1.5 – Пример слова «книга» в арабском языке

Чтение-запись в арабском языке выполняется справа налево. Буквы могут быть изображены диакритическими символами строки. Текст, даже в этом случае, это последовательная запись звуков. Похожие проблемы оформления текстов можно встретить в других языках с письмом справа налево, например в иврите. Это, разумеется, создает дополнительные сложности для индексирования

Затем осуществляется классификация с помощью методов машинного обучения. При этом сначала обрабатываются различные форматы файлов, потом выявляется кодировка и язык. При этом сохраняется информация о разметке текста.

Следует отметить, что при обработке ошибок форматов, например к «суп тегов» («tag soup) большинство систем ведут себя толерантно.

Далее для индексирования необходимо ввыделить определяется структурную единицу документа (document unit).

Уровни детализации зависят от многих факторов и ее выбор определяется системой информационного поиска. Специалист, разрабатывающий или внедряющий систему должен хорошее представлять содержание коллекции документов, понимать потребности пользователей.

Допустим, элемент индексирования подходящего размера выбран. Теперь в текст выделяют лексемы и одновременно удаляют некоторые символы, например знаки пунктуации.

Лексема (token) — некоторая конечная последовательность символов в документе, являющаяся семантической единицей [3,12].

Тип (type) —набор всех лексем, состоящих из одной и той же последовательности символов [3,12].

Термин (term) —тип (возможно, нормализованный), включенный в словарь системы информационного поиска[3,17.

Однако термины — это не лексемы в точно таком виде, в каком они встречаются в документе, поэтому к ним применяют различные процессы нормализации.

1.1.5 Вопросы к лекции № 1

1 Каковы основные понятия информационного поиска и обработки естественного языка?
2 Назовите и коротко опишите модели информационного поиска
3 Отобразите схематически и поясните логику булевской модели (теория множеств и булева алгебра) информационного поиска и обработки естественного.
4 Отобразите схематически и поясните логику векторной модели (векторные пространства и линейная алгебра) информационного поиска и обработки естественного языка.
5 Отобразите схематически и поясните логику вероятностной модели (множества, теория вероятностей) информационного поиска и обработки естественного языка.
6 Опишите технологию обработки булевых запросов.
7 Сравните информационный поиск, реализующий расширенную булевскую модель и ранжированный поиск.
8 Дайте определение понятий лексикон терминов, список слово позиций, стоп-слова, токен, терм, парсинг, лексема, стемминг.
9 Что такое токенизация и зачем она нужна?

1.2 Лекция №2 Словари и нечёткий поиск

План лекции

— поисковые структуры для словарей
— запросы с джокером
— исправление опечаток
— фонетические исправления
— Построение индекса:
  - – блочное индексирование, основанное на сортировке
  - – однопроходное индексирование в оперативной памяти
  - – распределённое индексирование
  - – динамическое индексирование
  - – другие типы индексов
— Сжатие индекса:
  - – статистические характеристики терминов в информационном поиске
  - – сжатие словаря
  - – сжатие инвертированного файла

1.2.1 Поисковые структуры для словарей

Главными причинами, при которых качество анализа становится сильно падает либо его проведение становится проблематичным является некорректно оформленная информация и различные ошибки в данных.

Чтобы устранить подобные проблемы применяется процедуры улучшения данных. Рассмотрим основные из них.

Стандартизация –приведение к единому формату данных иными словами- унификация [3,12].

Парсинг – грамматический или лексический анализ текста [3,17. Производится деление поля на атомарные значения (рис. 2.1).

Рисунок 2.1 – Парсинг данных

Для стандартизации служат **машинные словари** (рис. 2.2).

Рисунок 2.2 – Применение машинных словарей

Регулярные выражения (англ. regular expressions) — формальный язык поиска и осуществления манипуляций с подстроками в тексте, основанный на использовании метасимволов (символов-джокеров, англ. wildcard characters) [3,12]. Для этого служит строка-образец (англ. pattern, рус. шаблон, маска), которая состоит из метасимволов и задается правило поиска. Также задаётся строка замены, которая может содержать в себе специальные символы и предназначена для манипуляций с текстом.

Очистка данных – служит для выявления и исправления ошибок. Методика проведения очистки различна. Метод **частоты появления** основан на анализе заданного значения или его комбинаций в данных (рис. 2.3).

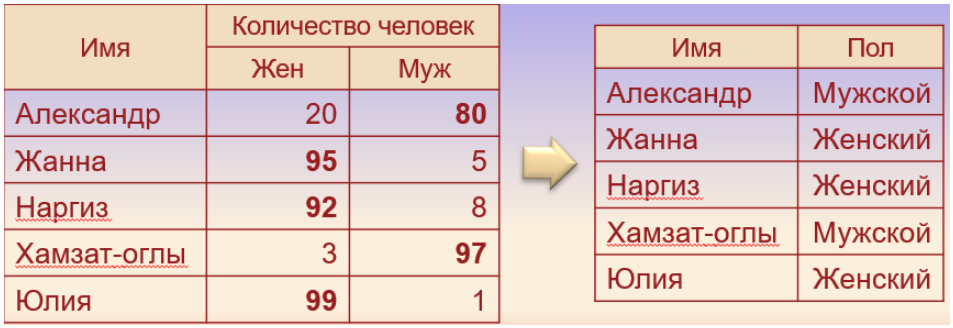


Рисунок 2.3 – Схема очистки по методу частоты появления

Алгоритма **контрольных чисел** использует расчет определенных функций, которые служат для контроля правильности номеров банковских карти различных идентификаторов и т.д. (рис. 2.4).

Рисунок 2.4 – Схема очистки по методу контрольных чисел

Анализ строк проверяет схожесть записей по алгоритмам сравнения значений - метода Левенштейна и др. (рис. 2.5).

Рисунок 2.5 – Схема очистки по методу анализ строк

Дедубликация - поиск совпадающих и похожих объектов служит для устранения повторов.

Кроме того методы очистки данных включают:

— наложение определенных правил на контролируемые поля;
— индексирование слов по звучанию;
— проверка по статистическим значениям;
— кластерный анализ.

Обогащение –насыщение данных новой информацией, которая позволяет сделать их более значимыми с точки зрения решения некоторой аналитической задачи.

Исследование взаимосвязанных объектов происходит на этапе **анализа связей,** здесь же и выявляют закономерности между ними (рис. 2.6).

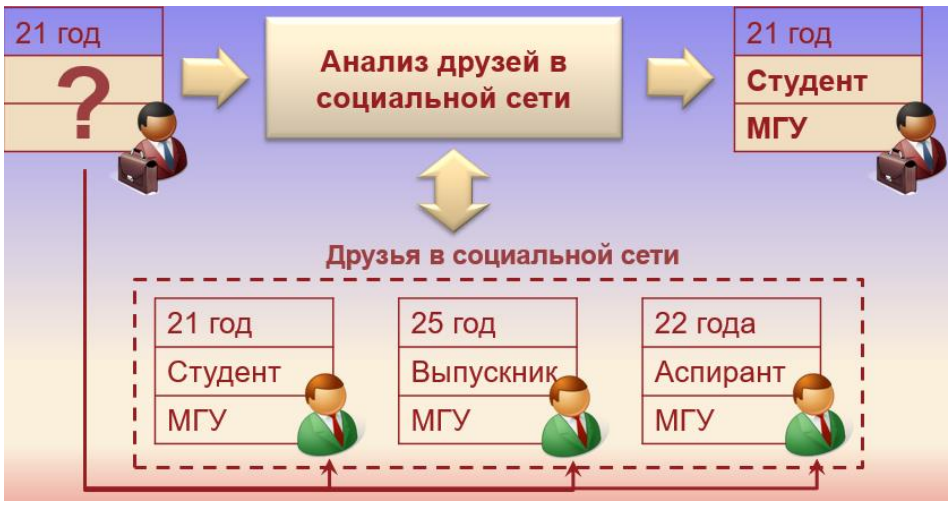


Рисунок 2.6 – Исследование взаимосвязанных объектов на этапе анализа связей

Как видно из рисунка 2.6 анализ связей позволяет в социальной сети определить возможные дружеские связи между выпускниками вуза.

Поиск близких объектов проверяет на «схожести» значений их признаков.

1.2.2 Запросы с джокером

Си́мвол-джо́кер (си́мвол подстано́вки) — символ, используемый для замены других символов или их последовательностей, приводя таким образом к символьным шаблонам. Развитием символов-джокеров являются регулярные выражения.

Запросы с джокером используются когда:

— Неизвестно, как правильно пишется термин

— Есть несколько вариантов написания

— Нужны документы, содержащие вариант термина, унифицированный в результате стемминга, но не известно, выполняет ли поисковик стемминг abc* - запросы с замыкающим джокером

Оптимальный словарь – двоичное дерево *abc – запросы с ведущим джокером Словарь – обратное двоичное дерево abc*def

Словарь – B-дерево и обратное B-дерево: получаем от каждого множество соответствующих началу и концу терминов, далее ищем пересечение. Необходимо только исключать случаи типа abc*abc

Запросы с джокером общего вида.

Алгоритм выполнения запроса с джокером общего вида схематически можно схематически представить (рис.2.7)

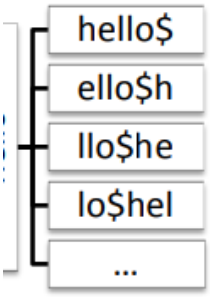


Рисунок 2.7 – Схема запроса с джокером общего вида

Перестановочный индекс (ПИ) $ - специальный символ, конец термина Чтобы построить ПИ, надо перебрать все варианты исходного термина с $ в конце, полученные циклической перестановкой символов. Набор терминов с перестановкой символов называется лексиконом перестановок (permuterm vocabulary).

Пусть есть запрос m*n.

1) Применяем циклическую перестановку: n$m*.

2) Ищем в перестановочном индексе такую строку

3) Восстанавливаем по перестановкам исходные термины

А как обработать запрос вида fi*mo*er?

1) Найдем все термины словаря, соотв. индексу er$fi*.

2) Полным перебором отфильтруем термины и оставим те, где есть символы mo.

Недостаток: словарь становится довольно большим.

К-граммный индекс для шаблонных запросов К-грамма – это последовательность, состоящая из К символов.

Пример 3-грамм для термина castle: $ca, cas, ast, stl, tle, le$, e$c. КГ индекс содержит все К-граммы, образованные из всех терминов лексикона. Каждый инвертированный список ставит в соответствие К-грамме все термины лексикона, её содержащие.

Пример: запрос re*ve

1)Переформируем запрос $re AND ve$

2)Поиск по 3-граммному индексу: relieve, remove…

3)Поиск соответствующих терминов в стандартном инвертированном индексе 4)Постфильтрация – сравнение с исходным шаблоном (важно если запрос red*; поиск по $re AND RED; retired)

1.2.3 Исправление опечаток

Многообразие программных продуктов по коррекции орфографии на рынке информационных услуг объясняется довольно просто. Это ответ специалистов на потребности пользователей мобильных устройств по набору текста на естественном языке.

Вместе с тем для русского языка качество автоматической обработки ввода и проверки орфографии еще идеально и эта сфера постоянно модернизируется. Действительно число текстов в социальных медиа (блогах, социальных сетях, форумах), далеких от нормативного языка постоянно растет,

круг создающих их пользователей неограничен. Естественно, при наборе текста возникают ошибки:

1) различные сокращения и неверная расстановка дефисов и тире порождают ошибки токенизации;

2) опечатки приводят к ошибкам при морфологическом и лемматизации текста, а также к ошибкам при синтаксическом анализе;

3) некорректное применение прописных букв препятствует автоматическому поиску имен и названий.

Перечисленные выше ошибки относятся к **опечаткам**, т.к. это ошибка набора в тексте, связанная с символами или написанием заглавных/строчных букв.

Программы обработки по исправлению опечаток бываю 2-х типов:

1) обнаружения (spell-checkers), указывают на опечатки;
2) исправления (spellcorrectors), правят найденные опечатки.

В функционал систем автоматической коррекция орфографии включены и другие возможности: развертывание сокращений, приведение к единому стилю написания кавычек и тире и т. д.

Большинство моделей коррекции орфографии основаны на использовании словаря со списком словоформ.

Разберем методику исправления опечаток в запросах. Например, необходимо найти документы, содержащие термин *carrot* , введя запрос *carot*. Т.е. введен один символ *r* и в слове *carrot* допущена опечатка.

Реализация исправления опечаток

В основе большинства алгоритмов исправления опечаток лежат два фундаментальных принципа.

1 Выбирается ближайший правильный способов написания искаженного запроса Меры близости а также алгоритмы ее эффективного вычисления рассмотрим чуть позже при изучении понятия «расстояние редактирования».

2. Если два правильно записанных запроса связаны (или почти связаны) друг с другом, то выбирается более распространенный вариант.

Например, запросы *grunt* и *grant* выглядят одинаково подходящими вариантами для исправления запроса grnt . Следовательно, алгоритм должен выбрать тот вариант, который чаще используется. Наиболее простой оценкой частоты использования слова является количество появлений этого термина в коллекции документов; следовательно, если слово *grunt* встречается чаще, чем слово *grant* , то следует выбрать именно его.

Во многих поисковых системах, особенно в вебе, используется другой способ оценки распространенности термина. Его идея заключается в том, чтобы использовать в качестве исправления то, что чаще всего встречается в запросах других пользователей. В частности, если слово *grunt* в запросах встречается чаще, чем слово *grant* , то , скорее всего, пользователь, напечатавший слово *grnt* хотел напечатать слово *grunt*. Методы исправления ошибок в запросе основаны на этих алгоритмах.

Функциональность исправления ошибок предоставляются пользователю одним из следующих способов:

1 В ответ на запрос *carot* всегда возвращаются документы, содержащие слово *carot* , а также все возможные «исправленные» варианты этого слова, включая *carrot* и *tarot* .

2. Аналогично п. 1, но только если слова *carot* нет в словаре.

3. Аналогично п. 1, но только если в ответ на оригинальный запрос система вернула слишком мало документов (например, меньше пяти).

4. Если количество документов, возвращеных в ответ на запрос, м е н ь ш е установленного порога, то поисковая система предлагает пользователю вариант исправления (spelling suggestion): это предложение состоит из исправленных терминов.

Таким образом, поисковая система в ответ н а запрос пользователя может спросить: «Вы имели в виду слово *carrot*?»

Методы исправления ошибок

Остановимся на двух методах исправления ошибок: исправлении изолированного термина (isolated-term correction) и исправлении с учетом контекста context-sensitive corection). При исправлении изолированного термина термины запроса исправляются по отдельности. Пример со словом **carot** иллюстрирует этот подход. Этот метод не позволяет, например, обнаружить, что запрос *flew form Heathrow* содержит искаженный термин *from* , поскольку каждый из терминов запроса по отдельности записан правильно.

Рассмотрим два метода решения задачи исправления изолированных терминов: расстояние редактирования и перекрытие k-грамм.

Расстояние редактирования

Расстояние редактирования между двумя строками символо в s1 и s2 — это минимальное количество операций редактирования (edit operations), с помощью которых строку s_1 можно трансформировать в строку s_2.

Операции редактирования, позволяющие это сделать, включают в себя следующие преобразования:

1) вставка символа в строку,
2) удаление символа из строки,
3) замена символа в строке другим символом.

Расстояние редактирования называется *расстоянием Тевеншейна* (Levenshtein distance).

Например, расстояние редактирования между словами *dog* и *cat* равно трем. Расстояние редактирования можно обобщить, если разным операциям редактирования присвоить разные веса.

Например, операции замены символа *s* символом *p* можно присвоить более высокий вес, чем операции замены символа *s* символом *a* (так как буква *a* ближе к букве *s* на клавиатуре). Присвоение весов с учетом правдоподобности замены на практик оказалось очень эффективным (проблема фонетической схожести).

Будем придерживаться допущения, что все операции редактирования имеют один и тот же вес. Хорошо известно, как вычислить расстояние редактирования между двумя строками за время $0(|s_1|x|s_2|)$, где $|s_i|$ — длина строки s_i. Для этого используется алгоритм динамического программирования, представленный на рис. 14, где строки s_1 и s_2 представлены в виде массивов. Этот алгоритм заполняет все (целочисленные) ячейки матрицы m, размеры которой равны длинам строк s_1 и s_2; после выполнения алгоритма элемент (i,j) содержит расстояние редактирования между строками, состоящими из первых i символов строки s_1 и первых j символов строки s_2. Основной шаг динамического программирования отображен в строках 8 - 10 (рис. 2.8), в которых вычисляется минимум трех величин с учетом замены символа в строке s_1, вставки символа в строку s_1 и вставки символа в строку s_2.

```
EditDistance(s1, s2)
 1  int m[|s1|, |s2|] = 0
 2  for i ← 1 to |s1|
 3  do m[i, 0] = i
 4  for j ← 1 to |s2|
 5  do m[0, j] = j
 6  for i ← 1 to |s1|
 7  do for j ← 1 to |s2|
 8        do m[i, j] = min{m[i–1,j–1] + if(s1[i] = s2[j]) then 0 else 1 fi,
 9                         m[i–1,j] +1,
10                         m[i,j–1] + 1}
11  return m[|s1|, |s2|]
```

Рисунок 2.8 -. Алгоритм динамического программирования для вычисления расстояния редактирования между строками S_1 и S_2

Матрица вычисления расстояния Левенштейна по алгоритму, представленна на рис. 2.9. Каждая ячейка разбита на четыре части размером 2X2, в которых записаны три числа, минимум которых дает четвертое число.

В правой нижней части записывается минимум чисел, соответствующих шагу динамического программирования в алгоритме , представленном на рис. 2.8. Остальные три ячейки содержат числа $m[i - 1,j - 1]+0$ или 1 в зависимости от того, выполняется ли условие $s_1[i] = s_2[j]$, а также числа $m[i - 1,j] + 1$ и $m[i,j - 1] + 1$.

Ячейки, определяющие расстояние редактирования в данном примере, выделены курсивом.

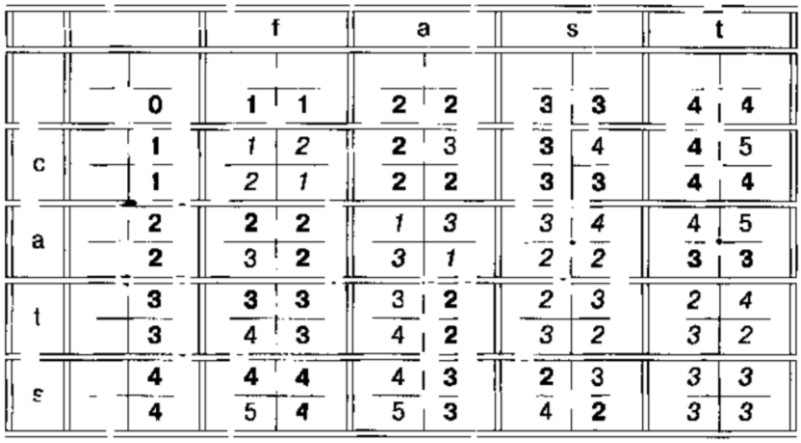


Рисунок 2.9 - Пример вычисления расстояния Левенштейна.

Для исправления опечаток недостаточно уметь вычислять расстояние редактирования для заданных множества строк *V* (соответствующих терминам в лексиконе) и строки запроса *q* необходимо найти строку или строки из множества *V* с минимальным расстоянием редактирования до запроса *q*. Эту проблему можно рассматривать как задачу декодирования, в которой кодовые слова (строки из множества *V)* заданы заранее. Очевидны й способ решения - вычислить расстояние редактирования от запроса *q* до каждой из строк, принадлежащих множеству *V,* а затем выбрать строку или строки с минимальным расстоянием. Этот исчерпывающий поиск является чрезмерно затратным. По этой причине для эффективного поиска терминов в словаре, имеющих небольшое расстояние редактирования до терминов запроса, на практике применяется большое количество эвристических правил.

Альтернативной эвристикой в методе расстояния редактирования является метод скелетов (skeleton), заключающийся в выделении всех скелетов — подпоследовательностей букв из терминов индекса и запроса, таких, что их длина меньше исходного термина на заданный порог (1 или 2 буквы). Хеширование скелетов терминов словаря и хеш-поиск всех скелетов термина из запроса дают за линейное время полный список кандидатов на коррекцию.

К-граммные индексы для исправления опечаток

Для того что бы еще больше ограничить множество терминов лексикона, для которых вычисляется расстояние редактирования до терминов из запроса, можно использовать k-граммный индекс, описанный в разделе. Этот индекс позволяет найти в лексиконе термины с небольшим расстоянием редактирования до запроса *q*. Найдя эти термины, м ы м о ж е м определить среди них термины с минимальным расстоянием редактирования до запроса *q*.

Фактически k-граммный индекс используется для поиска терминов лексикона, содержащих большое количество k-грамм, общих с запросом. Идея заключается в том , что при разумной трактовке выражения «большое количество общих k-грамм» процесс поиска по существу сводится к однократному просмотру «словопозиций» для k-грамм, входящих в запрос *q*.

Например, на рисунке 2.106 показана часть слово позиций для трех биграмм в запросе *bord* . Если необходимо найти термины лексикона, содержащие по крайней мере две из них, то однократное сканирование записей позволило бы перечислить все такие термины.

В примере, показанном на рис. 2.10, перечислены термины aboard , boardroom и border.

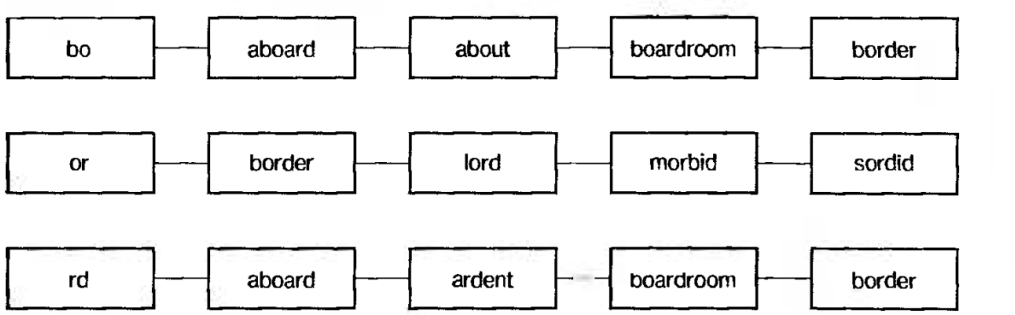


Рисунок 2.10 - Совпадение по крайней мере двух из трех биграмм в запросе bord

Применение «в лоб» пересечения инвертированных списков с помощью их последовательного просмотра сразу обнаруживает недостатки требования лишь присутствия в терминах лексикона фиксированного количества k-грамм из запроса q: при этом идентифицируются термины наподобие boardroom, представляющие собой «неправдоподобное» исправление слова bord. Следовательно, нужно предпринять дополнительные усилия для перекрытия k-грамм между термином лексикона и запросом q.

Применим коэффициент Жаккара (Jaccard coefficient), характеризующий перекрытие двух множеств A и B и равный $|A \cap B|/|A \cup B|$. Имеется множество Л-грамм в запросе q и множество k-грамм в термине лексикона. При сканировании осуществляется переход от одного термина лексикона *t* к следующему, при этом вычисляется коэффициент Жаккара для строк *q* и *t*.

Если коэффициент превышает пороговое значение, то термин добавляется в результаты; в противном случае, происходит переход к следующему термину в инвертированном списке.

Для того что бы вычислить коэффициент Жаккара, необходимо иметь множества k-грамм для строк *q* и *t*. Поскольку мы просматриваем «словопозиций» всех k-грамм в запросе *q*, т о сразу получаем все эти k-граммы в свое распоряжение. А как получить k-граммы для термина *t*? В принципе , можно было бы вы делить их все «на лету». На практике это не только не эффективно, но и нецелесообразно ; в подавляющем большинстве случаев сами записи содержат не саму строку *t*, а лишь ее идентификатор. Следует подчеркнуть , что для вычисления коэффициента Жаккара необходимо знать лишь длину строки *t*.

Коэффициент Жаккара можно заменить другими показателями, допускающими эффективное вычисление "на лету" в ходе сканирования записей.

Использовать их для исправления опечаток можно различными методами.

Один из эмпирических методов заключается в следующем: сначала для перечисления множества кандидатов среди терминов лексикона, представляющих собой потенциальные исправления запроса *q*, используется k-граммный индекс.

Затем необходимо вычислить расстояние редактирования между запросом *q* и каждым термином множества.

И наконец из множества выбираются термины с небольшим расстоянием редактирования от запроса *q*.

Исправление опечаток с учетом контекста

Методы исправления ошибок в некоторых случаях не применимы «в лоб», например, в запросе *flew form Heathrow*, здесь визуально все три термина написаны правильно.

Хотя при ответе на подобную фразу, подобную этой допустим, будет возвращено НЕЗНАЧИТЕЛЬНОЕ число документов, тогда поисковая система может принять решение исправить этот запрос на *flew from Heathrow*.

Предложить исправления для каждого из трех терминов в отдельности и собрать их в фразу.

В примере *flew form Heathrow* перечисление должно содержать фразы наподобие *fled from Heathrow* и *flew fore Heathrow*. Для каждой такой фразы поисковая система выполняет запрос и определяет количество соотвегствий. Если исправлений отдельных терминов будет слишком много, то такое перечисление может оказаться очень затратным. Так, при создании альтернатив для терминов *flew* и *form* оставим только те комбинации, которые чаще остальных встречаются в коллекции документов или в логах запросов, в которых хранятся предыдущие запросы пользователей.

Например , мы можем оставить словосочетание *flew from* как возможную альтернативу и попробовать расширить ее до трехсловного запроса, проигнорировав варианты *fled fore* и *flea form*. В данном случае словосочетание из двух слов filed fore является менее вероятным, ч ем сочетание flew from.

Затем достаточно просто расширить список наиболее часто встречающихся словосочетаний из двух слов (например, *flew from*) с помощью исправлений термина *Heathrow*. В качестве альтернативы использованию статистики биграмм в коллекции можно использовать лог запросов, заданных пользователями; разумеется, он может содержать в том числе запросы с ошибками.

1.2.4 Фонетические исправления

Последний метод, используемый пр и нечетком поиске, предназначен д л я *фонетических исправлений* (phonetic correction), т.е. ошибок, возникающих, когда пользователь за писывает запрос так, как он его слышит. Такие алгоритмы особенно полезны для поиска имен людей. Метод основан на генерировании "фонетического хеша" для каждого термина. Таким образом, звучащим одинаково терминам ставится в соответствие одно и тоже число.

Эта идея возникла в начале XX века в международных отделах полиции, перед которым и стояла задача поиска преступников, не обращая внимания на разное написание этих имен в разных странах. В основном этот метод используется для исправления фонетических ошибок в собственных именах .

Алгоритмы фонетического хеширования называют tot алгоритмами **Soundex**. Существует оригинальный алгоритм Soundex, имеющий несколько вариантов. Его схема выглядит так:

1. Каждый индексируемый термин преобразуется в четырехсимвольную сокращенную форму.
2. На основании сокращенных форм строится инвертированный индекс для поиска исходных терминов soundex-индекс).

2. Делается тоже самое для терминов запроса.

3. Если поступает запрос на сравнение строк по звучанию, выполняется поиск по индексу Soundex.

Варианты реализации алгоритмов Sounde связаны с методом преобразования терминов в четырехсимвольное представление. В результате применения наиболее распространенного метода возникает четырехсимвольный код, в котором первый символ — это буква алфавита, а остальные три — цифры от 0 д о 9.

1 Запишем первую букву термина.

2. Заменим все буквы A, E, I, O, U, H, W и Y н у л е м 'O'.

3. Заменим буквы цифрами следующим образом.

4 . B , F , P , V H a l .

5. C, G, J, K , Q , S, X , Z на 2.

6. D , T на 3.

7. 1 н а 4 .

8. M , N на 5.

9. Я н а 6 .

10. Циклически удали м одну из каждой п а р ы соседних одинаковых цифр.

11. Из полученной строки удалим все нули. Дополним результат замыкающим и нулями и возвратим первые четыре позиции, представляющие собой букву, за которой следуют три цифры.

В качестве примера укажем, что слово Herman отображается в код H655. Получив запрос (скажем, *herman*) , мы вычисляем его soundex-код, а затем находим в лексиконе все термины , соответствующие этому soundex-коду в soundex-индексе. Теперь выполним поиск результирующего запроса в стандартном обратном индексе.

Этот алгоритм основан на нескольких наблюдениях:

1) гласные могут заменять друг друга в транскрипции имен;

2) согласные с подобными звуками (например, D и T) относятся к одному классу эквивалентности.

Благодаря этому похожие по звучанию имена могут иметь одинаковые soundex-коды. Эти правила зависят от системы письменности . Например ,

китайские имена могут быть за писаны как в транскрипции Вейда - Джайлса (Wade - Giles) , так и в транскрипции пиньинь (Pinyin). Несмотря на то что алгоритмы Soundex с не которыми изменениями работают с обеими транскрипциями — например, термин hs в транскрипции Вейда - Джайлса и терми н x в транскрипции пиньинь переводятся в один и тот же код 2, — в некоторых случаях происходит сбой: например, символ j в транскрипции Вейда - Джайлса и символ α в транскрипции пиньинь переводятся в разны коды.

1.2.5 Построение индекса

Процесс индексирования можно разбить на пять шагов:

1) анализ структуры – выделение заголовков, абзацев и т.п.; удаление html-разметки и т.д;

2) токенизация – разбиение текста на слова, удаление знаков препинания;

3) удаление стоп-слов - высокочастотных служебных слов (предлогов, союзов и т.п.);

4) лемматизация – приведение слов к нормальной (например, словарной) форме;

5) взвешивание – частота слов в документе.

Главная задача DR – это выбрать из коллекции документы релевантные запросу и только их. И, возможно, упорядочить выборку по релевантности.

Релевантность – мера близости документа и запроса.

Существующие подходы к решению задачи DR:

— индексирование по ключевым словам;

— двоичный поиск;

— ранжированный поиск (vector-space model);

— вероятностная модель.

Индексирование по ключевым словам.

— Информационный поиск начинается не с написания запроса, а с индексирования документов;

— каждому документу сопоставляется поисковый образ документа (ПОД). Пример: документ – книга, ПОД – алфавитный указатель в конце книги.

Двоичный поиск.

— Поисковый образ запроса (ПОЗ) – формула. Термины и логические связки (AND, OR, NOT);

— ПОЗ можно рассматривать как двоичную маску;

— способ наложения маски: каждому операнду сопоставляется подмножество документов, затем над множествами выполняются соответствующие операции (пересечение, объединение, дополнение);

— ровно 2 значения релевантности: true, false.

Ранжированный поиск (vector space model).

— ПОД и ПОЗ представляют собой векторы в пространстве терминов;

— значения элементов векторов задаются некоторой функцией (наиболее популярна tf*idf);

— релевантность – близость векторов поисковый образ документа и запроса в пространстве терминов (косинус угла между ними).

Вероятностная модель.

— Базируется на расчете вероятности того, что документ релевантен запросу;

— делается ряд допущений: документ либо релевантен запросу, либо нет, термины распределены по документам коллекции независимо, релевантность одного документа не зависит от других;

— в целом качество поиска не лучше, чем у модели vector space.

1.2.6 Сжатие индекса

Поговорим о сжатии словаря и инвертированного списка для коллекции документов. Зачем вообще сжимать обратный индекс? Если говорить про словарь, то мы хотим сделать его настолько маленьким, чтобы он помещался в оперативную память. При этом мы хотим, чтобы в оперативную память помещалось что-то еще. Если мы говорим про инвертированный список, то, во-первых, это экономия места на диске; во-вторых, это уменьшение времени чтения списков жесткого диска, и большинство поисковых систем сейчас держат значительную часть инвертированного списка в памяти, поэтому нужно сделать его как можно меньше.

Рассмотрим корпус документов на примере дампа русскоязычной Википедии, который вы можете сами скачать. Всего у нас 2.800.000 документов, при этом среднее число лемм в документе равняется 250. При этом у нас около 2,5 миллионов термов, и длина каждого терма в среднем 8,5 байта. Давайте посмотрим, каких эффектов мы можем добиться всего лишь предварительной обработкой текста. Если мы удалим все числа из документов, то мы сократим размер словаря на 2%, а размер инвертированного списка на 9%.

При этом приведение к одному регистру позволит нам добиться сокращения словаря в сумме до 19%, а удаление топ-30 слов поможет сократить нам инвертированные списки в сумме до 38%. При этом если мы еще применим стемминг, то размер инвертированных списков всего у нас уменьшится до 52%.

Поговорим о законе Хипса ($M = kT^b$), который позволяет нам установить связь между размером словаря и числом лексем в коллекции. Размер коллекции M = K (это коэффициент) умноженное на T (это число лексем в коллекции) в степени b. Типичные значения для параметров k и b следующие: k находится в интервале от 30 до 100, а b равняется примерно 0,5.

Для русскоязычной Википедии метод наименьших квадратов дает следующие значения для оценки параметров M ($\log_{10}M = 0.52 \log_{10}T + 1.78$). Таким образом, параметр k у нас равен приблизительно 61, а параметр b равен приблизительно 0,52. Википедия — это хорошо структурированный, редактируемый набор документов. Поэтому там слова с ошибками и какие-то придуманные слова встречаются довольно редко. Но для той же коллекции документов из Интернета значение параметра b может достигать единицы,

потому что там редактирование отсутствует совершенно и много слов придуманных или взятых откуда-то еще. Если закон Хипса оценивает размер словаря в коллекции, то закон Ципфа позволяет получить оценку для относительных частот терминов. Он гласит следующее: i-тый наиболее частый термин имеет частоту проявления, пропорциональную 1 / i. Этот закон следует из следующих соображений. В языке очень немного терминов с большой частотой и очень много терминов с маленькой частотой. Таким образом, частота термина приблизительно равняется 1 / i либо k / i, где k — это какая-то константа нормализации.

Словарь необходимо сжимать, так как поиск начинается со словаря, и мы хотим держать весь словарь в памяти. Тогда размер словаря должен быть небольшим. Даже если словарь не находится постоянно в памяти, его чтение не должно занимать много времени.

Первый подход к хранению словаря будет таким. Мы будем на каждый терм выделять фиксированное количество байт. Например, 20 байт на сам терм, четыре байта на его частоту и четыре байта на указатель инвертированного списка для данного терма. Тогда всего в сумме мы получим 28 байт на терм.

Недостатки данного метода:

Во-первых, много места тратится впустую, ведь средняя длина слова у нас получилась около девяти символов. А мы для каждого символа выделяем 20 байт. И множество слов в русском языке длиннее 20 символов, например, дезоксирибонуклеиновая или четырехсотпятидесятисемимиллиметровое орудие.

Исправить данную ситуацию можно следующим образом:

Мы можем хранить все термы в одной строке, а в таблице хранить лишь смещение в этой строке. Тогда указатель на начало следующего слова будет как раз показывать нам границу текущего слова. И таким образом мы можем сократить размер словаря до 60%.

Посчитаем занимаемое место. Нам нужно четыре байта на частоту терма, четыре байта на указатель на инвертированный список и три байта на указатель на термин. В среднем нам нужно девять байт на термин в строке. Тогда всего нам нужно 20 байт, и мы можем сократить размер словаря с 65 мегабайт до 47 мегабайт.

Пойдем дальше и будем хранить указатель на терм в строке не k раз, а через какой-то промежуток. Тогда нам дополнительно нужно будет хранить длину терма в строке, но за счет того, что у нас указателей на сам терм будет меньше, мы добьемся какого-то сокращения.

Дальнейшее развитие идеи таково, что слова в словаре идут друг за другом в алфавитном порядке, то есть слова с одинаковыми префиксами идут друг за другом. Мы можем дополнительно кодировать префикс слова каким-то специальным словом, а затем вставлять этот префикс с помощью какого-то другого специального символа.

Подведем итог. Мы обсудили сжатие с потерями и без потерь, поговорили про законы Хипса и Ципфа и обсудили сжатие словаря: с

использованием непрерывной строки термов, с использованием блоков и с использованием фронтальной упаковки.

1.2.7 Вопросы к лекции №2

1 Опишите поисковые структуры для словарей. Каким образом словари способствуют организации нечёткого поиска.
2 – –Как составить и реализовать запросы с джокером?
3 Как произвести исправление опечаток в ходе информационного поиска и обработки естественного языка?
4 Как произвести фонетические исправления в ходе информационного поиска и обработки естественного языка?
5 Назовите и опишите основные методы индексирования.
6 Опишите процесс сжатия индекса.
7 Назовите статистические характеристики терминов в информационном поиске.
8 Назовите основные способы сжатия словаря и инвертированного файла?
9 Назовите основные компоненты информационно-поисковой системы.

1.3 Лекция № 3 Ранжирование, взвешивание терминов и модель векторного пространства

План лекции

Модель векторного пространства:
– параметрические и зонные индексы;
– частота термина и взвешивание;
– модель векторного пространства для ранжирования;
– варианты функций tf-idf.
Ранжирование в полнофункциональной поисковой системе:
– эффективное ранжирование
– компоненты информационно-поисковой системы
– влияние операторов языка запросов на ранжирование в векторном пространстве.

1.3.1 Модель векторного пространства.

Формальные составляющие модели информационного поиска:

— Логическое представление документов
— Логическое представление запросов
— Framework моделирования представлений документов и запросов, их связей
— Ранжирующая функция

Различают следующие классы моделей информационного поиска:

— Булевская модель
— Векторная модель
— Вероятностная модель

Модели векторного пространства или векторные модели также подвержены определенной градации:

— Обобщенная векторная модель (учет корреляции между термами)
— Латентно-семантический анализ (отображение документов и запросов в пространство концепций)
— Нейронные сети (нейроны —термы документов и документы, многошаговое распространение сигнала)

Обобщенная векторная модель (учет корреляции между термами) Generalized vector - несмотря на естественность сделанных предположений, на данным момент не понятно, когда эта модель показывает результаты лучше стандартной (практические эксперименты не показывают заметного превосходства). Минус модели - значительно более высокая трудоемкость

Латентно-семантический анализ (отображение документов и запросов в пространство концепций) - LSA – Latent Semantic Analysys основана на разложении по сингулярным значениям. Значительно снижает размерность используемого пространства. Утверждается что за счет отсева случайных шумов повышается качество поиска. Возможно нахождение документов не

прямо относящихся к запросу (т.е. не использующих термы из запроса, а только их синонимы, т.п.)

Нейронные сети (нейроны —термы документов и документы, многошаговое распространение сигнала) - Neural Nets Первоначальный запрос задает воздействие на нейроны термов документа, сигнал распространяется на документы, оттуда возможно опять на термы. Сила сигнала определяется весами термов (и их значимостью) Оценка релевантности – итоговый суммарный сигнал пришедший на документ позволяют найти документы, не прямо относящиеся к термам запроса. Также, как и LSA, может возвращать не прямо релевантные документы. На данный момент метод не тестировали на больших коллекциях

1.3.2 Параметрические и зонные индексы

На предыдущей лекции мы упоминали законы Хипса и Ципфа, теперь рассмотрим их подробнее.

Закон Г.С. Хипса (H.S. Heaps) описывает взаимосвязи словаря уникальных слов и объема документа., т.е.:

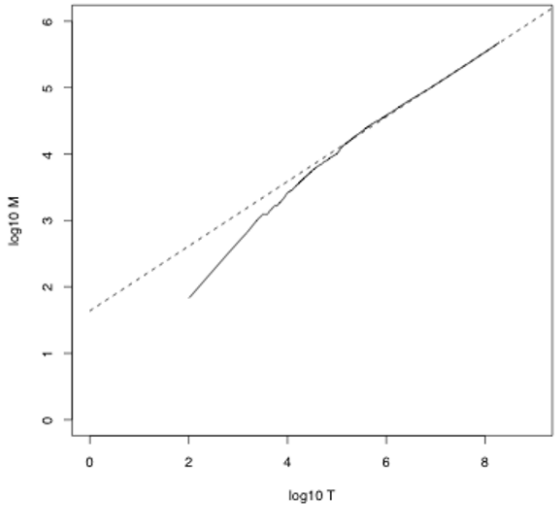
$$M(T)=\alpha T^{\beta} \qquad (3.1),$$

где M — это объем словаря уникальных слов, составленный из текста;

T лексемы в этом тексте;

α и β – определенные эмпирически параметры.

Для европейских языков α принимает значение от 10 до 100, а β - от 0.4 до 0.6. Иллюстрация закона представлена на рисунке 3.1.



Пунктирная линия построена по методу наименьших квадратов

$\log_{10} M = 0.49 * \log_{10} T + 1.64$

Получаем:

$M = 10^{1.64} T^{0.49}$

$k = 10^{1.64} \approx 44$

$b = 0.49.$

Рисунок 3.1 – График зависимости объема документа и словаря уникальных слов.

1.3.3 Частота термина и взвешивание

Известно, что короткие слова встречаются в текстах любого языка чаще, чем длинные. Оба закона Ципфа основаны на основе этого факта.

Закон1 - описывает связь частоты вхождения слова с рангом частоты. Наиболее часто встречающимся словам присваивается ранг *r=1*. Тем словам,

что встречаются реже – ранг, *r=2* и т.п. Кроме того, произведение частоты вхождения слова на ранг постоянная величина, заметил Ципф. Эта зависимость в виде гиперболы отображена на рис.3.2. Константа Ципфа принимает различное значение, но для одной языковой группы она постоянная величина. Частота вхождения слова является отношением количества появления слова к общему количеству слов в тексте. Значит, частота слова не может быть больше единицы, на практике получается диапазон значений 0,01-0,009.

Закон2 - частота вхождения слов зависит от количества слов, входящих в текст с этой частотой. График кривой этой взаимосвязи будет сохранять свои параметры для всех текстов написанных на одном языке. С другой стороны, на каком бы языке текст ни был написан, форма кривой Ципфа останется неизменной. Различными могут быть разве что коэффициенты.

Универсальность законов Ципфа порождает их широкое приложение и не только к текстам. Характеристики популярности ресурсов интернета, зависимость между количеством городов и числом проживающих в них жителей отвечают законам Ципфа. На рисунке 3.2 представлена графически гипербола связи частоты вхождения слова с рангом частоты кривая Ципфа.

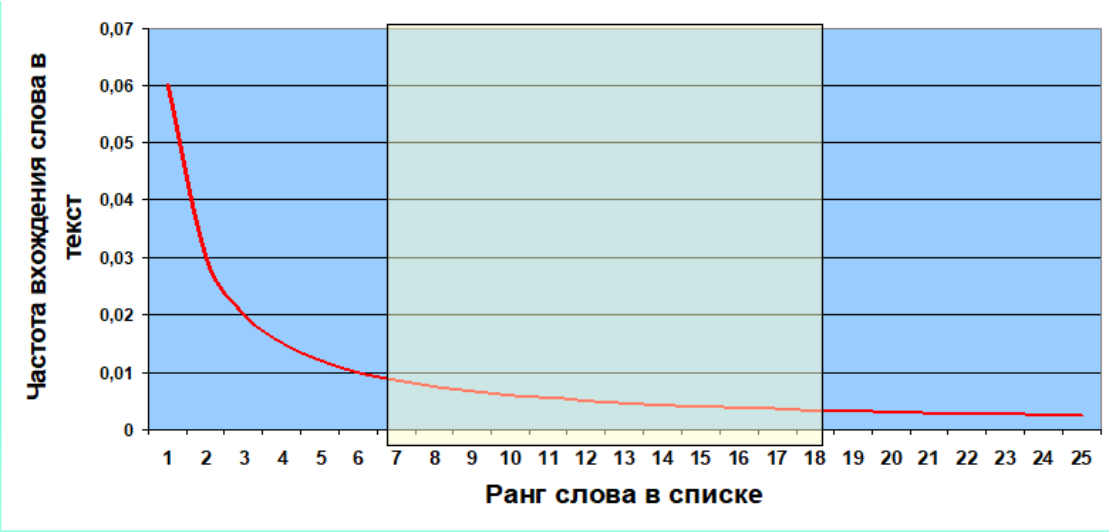


Рисунок 3.2 – График связи частоты вхождения слова с рангом частоты (Кривая Ципфа)

На рисунке 3.2 можно видеть, что **наиболее значимые для текста** слова лежат в средней части гиперболы. Наиболее характерные термины для текста содержит именно эта зона графика. Эти термины выражают содержание текста. С лева часто встречаемые слова —предлоги, местоимения, артикли и т.д. Редко встречаемые слова, которые не влияют особенно на смысл, соответственно справа.

Алгоритм поиска **значимых слов** может быть различным в современных поисковых системах. Тут важен их баланс полезности. Для того, чтобы этого достичь и избежать «шума», создается словарь «бесполезных» слов или «**стоп-**

слов». В своеобразный «стоп-лист» включают обычно артикли и предлоги, частицы и личные местоимения.

Выявлению значимости служит «**весовой коэффициент**». Слово, встречаемое редко имеет весовой коэффициент высокий коэффициент. Часто встречаемое слово, близкий к нулю. Параметр, определяющий «весовой коэффициент», называется **инверсная частота термина**. Система может учитывать морфологические особенности, местоположения слова при вычислении «весового коэффициента».

Большинство современных ПС основаны пространственно-векторной модели. Размещение в виртуальном многомерном пространстве всех документов базы позволяет получить результат, отвечающий запросу даже в том случае, когда в найденном документе не окажется ни одного ключевого слова.

1.3.4 Модель векторного пространства для ранжирования

В основе самых популярных методов ранжирования документов относительно запроса лежит векторная модель (vector space model, VSM). Следует заметить, что это не единственная модель, но в силу вышеупомянутого в рамках нашей дисциплины именно этой модели мы уделим внимание.

Предложенная в 1975 году (Salton 1975) модель векторного пространства, относится к алгебраическому типу. Термы, документа, отображаются на n-мерное линейное пространство.

Например, набор документов на очень ограниченном языке, состоящем только из двух слов: *hockey* и *cycling* (*хоккей* и велосипед) можно изобразить на графике (рис. 3.3). Документ, содержащий оба слова, будет стрелкой (вектором, или вектором термов), наклоненной под углом 45 градусов к каждой оси (рис. 3.3).

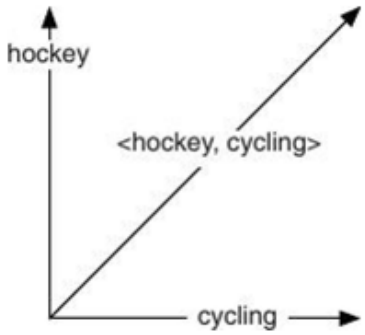


Рисунок 3.3. Графическое представление векторной модели для документа из двух слов: hockey и cycling

На рисунке представлен простейший случай, надо обобщить модель для большего числа измерений. Можно представить все множество документов в виде векторов в «-мерном линейном пространстве, причем каждому слову, встречающемуся хотя бы в одном документе, соответствует измерение.

Эти документы можно представить в векторном пространстве, пронумеровав уникальные слова. При этом, числа соответствуют осям векторного пространства. На рисунке 3.4 в 10-мерном пространстве илюсстация двух документов- векторов.

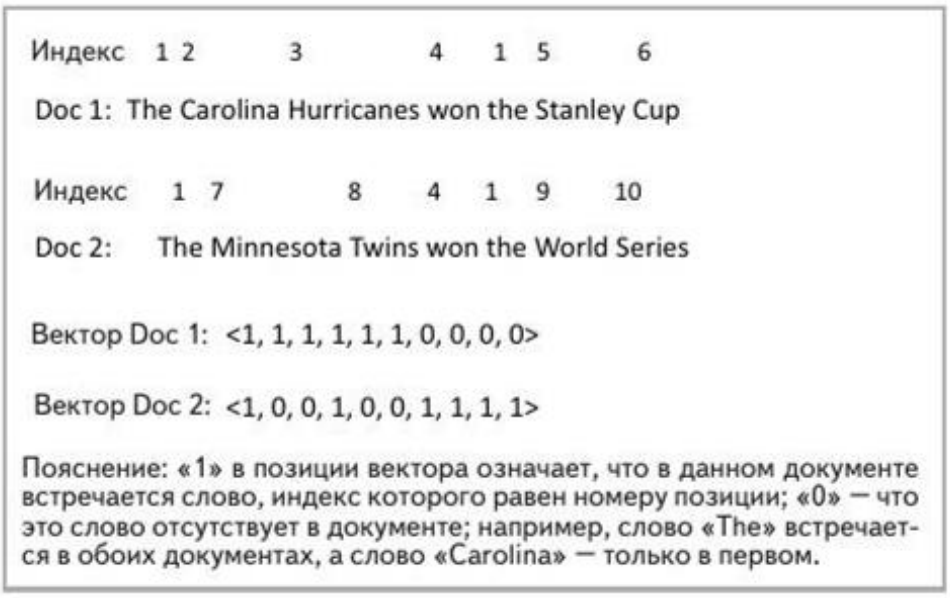


Рисунок 3.4. Векторное представление 2 документов в 10-мерном пространстве

Из рисунка 3.4 видно, что если у двух документов есть общее слово, то они пересекаются по соответствующей оси.

Естественно, модель (рис3.4) примитивна в реальности ПС обрабатывают огромное количество документов. В системе запоминается только наличие терма, но не его отсутствие, так устроен и инвертированный индекс, поэтому не нужно хранить множество нулей, ведь в большинстве документов встречается лишь малая часть всех слов. Современные поисковые системы хранят не просто 1, как обозначение наличия слова, но и его вес, как о характеристику важности терма (взвешенный вектор). Если сравнить термы запроса с термами и их весами в документах, где встречаются термы запроса, то можно будет вывести формулу, релевантности запроса.

Рассмотрим пример (рис. 3.4): артикль *the* встречается дважды в каждом из представленных на рисунке документов. Частота вхождения его 4, вес 2/4 = 0,5. Для слова *Carolina* вес 1/1 = 1. Подобным образом можно вычислить веса всех термов, получаем комплект взвешенных векторов. Для 1 документа вектор: **(**0.5**,** 1**,** 1**,** 0.5**,** 1**,** 1**,** 0**,** 0**,** 0**,** 0**)**

Перейдем к рассмотрению механизма сопоставления запросов с документами. Запросы, как и документы можно также отобразить на векторном пространстве (рис. 3.5). Если совместить начальные точки вектора запроса и вектора документа, то между ними образуется угол (рис. 3.5). Косинус этого угла - число от -1 до 1, это и есть ранг документа относительно запроса. Из рисунка 3.5 видно, что если угол между двумя векторами равен 0, то документ и запрос совпадают. Известно, *cos0’ =1*, ‘это соответствует логике

ранжирования (рис. 3.5). Выполнив подобные операции со всеми документов в наборе, получаем ранжированный список, являющийся результатом поиска.

Реальные системы не оценивают все документы, а лишь те, в которых встречается один или несколько термов из запроса.

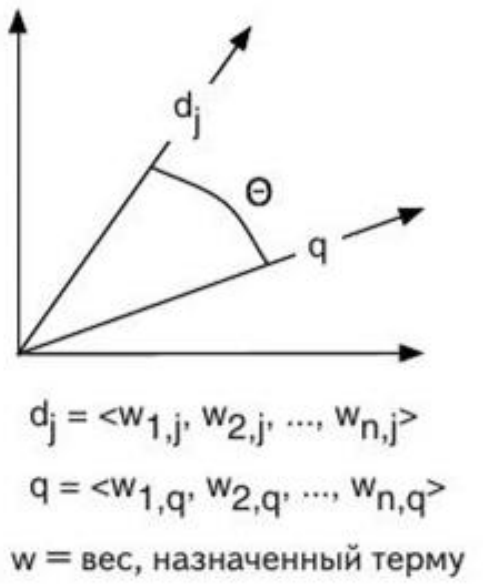


Рис. 3.5 - Сравнение запроса q с документом в векторной модели

Следует заметить, что на практике кроме оценки согласно модели У8М системы учитывают с различные характеристики: длина документа, средняя длина документов в наборе. Кроме того используются приемы назначения большего вес одному из документов или приписать разные веса разделам документа. Эти хитрости своеобразное ноу-хау разработчиков поисковых систем предпринимаются чтобы обеспечить релевантность результатов поиска.

Отображение результатов

При проектировании отображения результатов поиска важно определиться с вопросами:

— Кроме заглавия какие части документа еще следует отображать?

— Выделять ли слова из запроса?

— Исключать ли дубликаты?

— Какие инструменты предложить пользователю для удобства?

— Нужны ли средства расширения или сужения поиска?

Разберем на примере, пусть необходимо получить информацию по запросу *apple* в Google. Результат поиска представлен на рис. 3.6 заметим, что к первому результату система присовокупила самые популярные страницы сайта компании Apple и другие дополнительные данные.

На рисунке 3.6 показан по слову . Обратите внимание, что к первому результату Google добавляет ссылки на самые популярные страницы сайта компании Apple, биржевые котировки ее акций, адреса магазинов (даже карту), связанные с ней лица и другие данные. Далее (на рисунке не показаны) расположены ссылки и связанные поисковые запросы. Но на рис. 3.6 нет ни

одного результата о фрукте – яблоке. Это получилось в следствие высокого ранг популярной компании

Рисунок 3.6- Отображение результатов поиска по запросу *appl*

Так работают системы основанные на фасетном поиске, использующем кластеризацию Пользователь может уточнить или ограничить количество отображаемых результатов. Улучшить отображение результатов поиска позволяет автоматическая кластеризация (рис. 3.7).

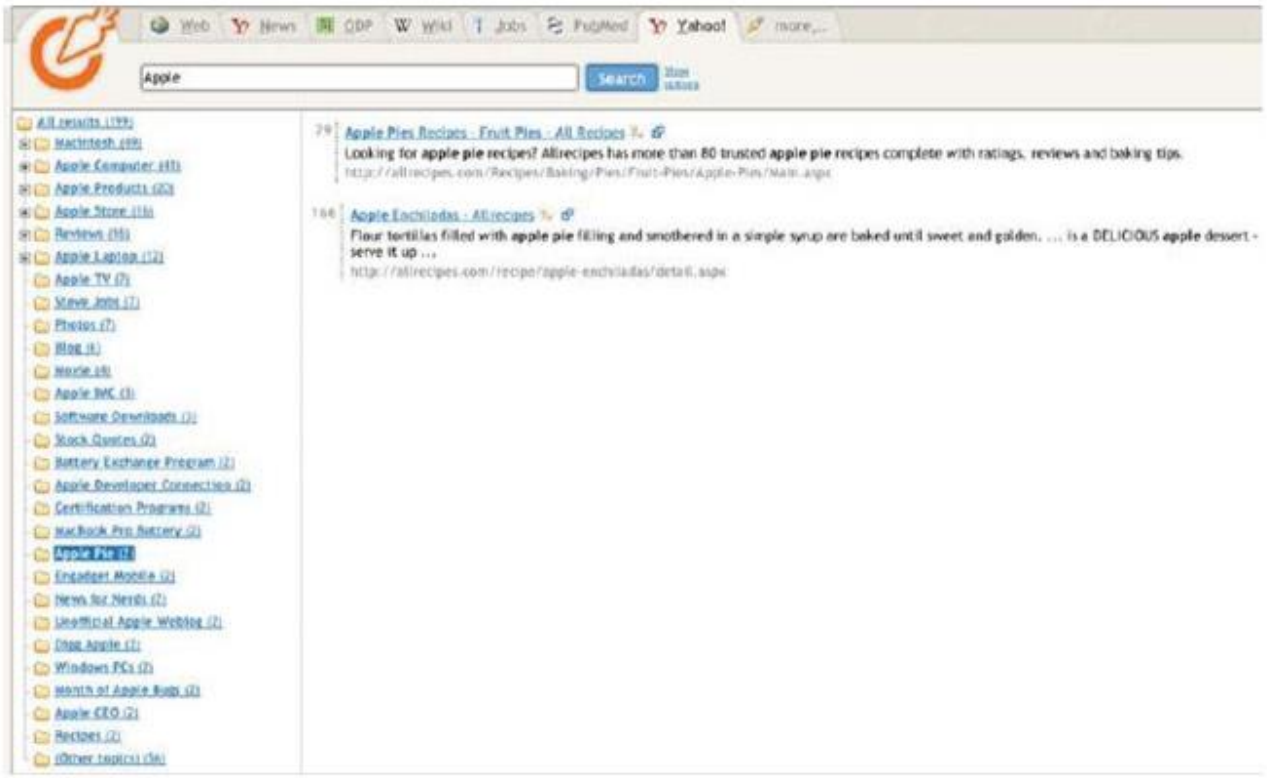


Рисунок 3.7. Отображение результатов в случае при неоднозначность запроса

Есть специальные сайты, предлагающие механизм кластеризации результатов, полученных от многих поисковых систем Carrot Search (http: / Avww.carrotsearch. com). На рисунке 3.7 в левой части экрана видны кластеры, выведенные на основе результатов.

1.3.5 Варианты функций tf-idf.

TF-IDF - от англ. TF — term frequency (частота слова), IDF — inverse document frequenc (обратная частота документа) — статистическая мера оценки важности слова в контексте документа.

Вес некоторого слова пропорционален количеству употребления этого слова в документе, и обратно пропорционален частоте употребления слова в других документах коллекции.

Эта мера часто используется в задачах анализа текстов как один из критериев релевантности документа поисковому запросу, при расчёте меры близости документов при кластеризации.

Структура формулы

TF — отношение числа вхождения некоторого слова к общему количеству слов документа. Таким образом, оценивается важность слова t_i в пределах отдельного документа.

$$\mathrm{tf}(t,d) = \frac{n_i}{\sum_k n_k},$$

Где n_i есть число вхождений слова в документ, а в знаменателе — общее число слов в данном документе.

IDF—— инверсия частоты, с которой некоторое слово встречается в документах коллекции.

Основоположником данной концепции является Карен Спарк Джонс. Учёт IDF уменьшает вес широкоупотребительных слов. Для каждого уникального слова в пределах конкретной коллекции документов существует только одно значение IDF.

$$\mathrm{idf}(t,D) = \log \frac{|D|}{|(d_i \supset t_i)|}, \qquad (3.1)$$

где

- |D| — количество документов в корпусе;
- $|(d_i \supset t_i)|$ — количество документов, в которых встречается *t* (когда *n≠0*).

Выбор основания логарифма в формуле не важен, так как изменение основания приводит к изменению веса каждого слова на постоянный множитель, что не влияет на соотношение весов.

Таким образом, мера TF-IDF является произведением двух сомножителей:

$$\text{tf-idf}(t,d,D) = \mathrm{tf}(t,d) \times \mathrm{idf}(t,D) \qquad (3.2)$$

Большой вес в TF-IDF имеют слова с высокой частотой употреблений в конкретном документа и с низкой частотой в других документах.

Применение

Показатель TF-IDF оценивает значимость слова в документе, на основе данных о всей коллекции документов. Данная мера определяет вес слова за величину пропорциональную частоте его вхождения в документ и обратно пропорциональную частоте его вхождения во всех документах коллекции.

Данный показатель используется для решения задач информационного поиска, в частности определения соответствия документа запросу и выявления семантически схожих документов.

Формулы, основанные на методе TF-IDF различны. Можно заметить, что используют разные коэффициенты, нормировки, логарифмические шкалы. Например, Яндекс долгое время использовала нормировку по самому «высокочастотном»у термину. Одной из наиболее популярных формул является формула Okapi BM25 (BM25).

Okapi BM25 — функция ранжирования, используемая поисковыми системами для упорядочивания документов по их релевантности данному поисковому запросу. Она основывается на вероятностной модели, разработанной в 1970-х и 1980-х годах Стивеном Робертсоном, Карен Спарк Джоунс и другими.

Сама функция носит название BM25 (BM от англ. *best match*), ее полное название с добавлением Okapi, в честь поисковой системы Лондонского городского университета, которой эта функция была впервые применена в 80-90х годах 20 века.

BM25 и ее модификации BM25F являются современные TF-IDF-подобными функциями ранжирования. Они широко используются в практике современных поисковых систем, в веб-поиске как компоненты более сложной машинно-обученной, функции ранжирования.

Например, если документ содержит 100 слов и слово «птица» встречается в нём 3 раза, то частота слова (TF) для слова «птица» в документе будет 0,03 (3/100). Вычислим IDF по формуле (3.1). Если «птица» содержится в 1000 документов из 10 000 000 документов, то IDF будет равной 4 (log(10000000/1000)). Для расчета окончательного значения веса слова необходимо TF умножить на IDF. В данном примере, TF-IDF вес для слова «птица» в выбранном документе будет 0,12 (0,03*4).

Значение TF-IDF часто используется для представления документов коллекции в виде числовых векторов, отражающих важность использования каждого слова из некоторого набора слов в каждом документе. Причем количество слов набора определяет размерность вектора. Эта модель называется векторной моделью. Она даёт возможность сравнивать тексты как вектора текстов в некоторой метрике.

1.3.6 Ранжирование документов

Ранжирование — это процесс сортировки веб-сайтов в представлении результатов поиска. Первостепенная задача процесса — определить сайты,

которые имеют наибольшую полезность для пользователя. Кроме того убрать из представления «неполезные» страницы и сайты с дублированным контентом.

Релевантные ссылки на сайты попадают в отображаемый результат в порядке определенном ранжированием.

При расчете релевантности будем учитывать:

— частоту вхождения в документ единичных слов запроса,

— совместную встречаемость слов

— взаимное положение.

Таким образом, вес документа по запросу в нашей системе складывается из трех составляющих:

$$W = kfWf + kpW\,p + kp5\;Wp5 \qquad (3.3)$$

где:

Wf – вес документа, вычисленный на основе TF*IDF алгоритма;

W p – вес документа, вычисленный на основе совместных вхождений в документ пар слов, расположенных рядом в запросе;

Wp5 – вес наиболее близкого к запросу пассажа документа;

kf kp kp5 – коэффициенты.

Очевидно для получения ненулевого веса в документе не обязательно присутствие всех слов запроса. Документы, для которых отношение суммарного IDF слов запроса, встречающихся в них, к суммарному IDF всех слов запроса превышает пороговое значение также попадают в число ранжируемых. Такие документы дополнительно «штрафуются» за отсутствующие слова, но вес некоторых из них в общем случае может даже превышать вес документов, содержащих все слова запроса.

Для подсчета TF*IDF используем модификацию стандартной BM25 формулы:

$$TF * IDF_{term} = \frac{f_{term} \times IDF_{term}}{f_{term} + k_1(b + L(1-b))} \qquad (3.4)$$

где: f_{term}– вес терма в документе, вычисленный на основе количества вхождений, с учетом ряда дополнительных факторов;

IDF_{term}– обратная частотность терма в коллекции, вычисленная по стандартной логарифмической формуле;

L – нормированная длина документа;

k_1, *b*– коэффициенты.

Общий *TF*IDF* вес документа получается суммированием полученных весов по всем термам запроса.

Отметим, что для документов, размер которых превышает константу k_2 (соответствующую в стандартной BM25 формуле средней длине документа в коллекции, а в нашей системе, задаваемой в настройках) вместо нормирования по длине используется метод разбиения документа на перекрывающиеся фрагменты. Это чтобы избежать занижения веса длинных документов, в которых имеется небольшой фрагмент с высокой релевантностью.

Фрагменты имеют фиксированный, задаваемый в настройках размер, меньший k_2, и берутся с наложением по всему тексту документа.

Вес каждого из фрагментов по каждому терму запроса оценивается по формуле (3.3) без нормирования по длине, то есть с $L=1$.

В результате, выбирается фрагмент документа, имеющий наибольший вес и его вес используется в качестве *Wf* веса документа в (3.3).

Для документов, имеющих длину меньшую, чем k_2, используется нормирование по длине по формуле

$$L = \frac{L_w + k_4}{k_3 + k_4} \tag{3.3}$$

где:

L_w – длина документа в словах;

k^3 k_4– коэффициенты, задаваемые в настройках.

Существенной особенностью TF*IDF ранжирования в описанном выше способе поиска является использование достаточно большого значения коэффициента в формуле (3.3), значительно большее обычно принятых.

Для прогона по веб-коллекции расстояние было выбрано как 2 (рядом или через одно) для случая, когда порядок слов в запросе и документе совпадает и 1 (только рядом) для случая, когда не совпадает.

В соответствие с этим условием вхождения слов обрабатываются по описанному выше TF*IDF алгоритму, отличается только набор коэффициентов.

Под **пассажем** мы понимаем фрагмент документа, размера, не превышающего заданный, в котором встречаются все термы запроса, либо значительная часть термов запроса, суммарный IDF которых превышает заданное ограничение.

При выборе лучшего пассажа документа основными факторами являются его полнота (наличие всех термов запроса), длина, порядок слов (его совпадение с порядком слов в документе), зона документа (заголовок, выделенный текст, обычный текст), в которой встретился пассаж, близость пассажа к началу документа. Учитывается также ряд дополнительных факторов.

Вес пассажа по каждому из факторов оценивается в баллах на основе специальных для каждого из них правил, после чего веса суммируются. Суммарный вес и будет весом пассажа. Из полученных весов пассажей выбирается максимальный для вычисления общего веса по формуле (3.3).

Проанализировать актуальность описанных выше алгоритмов и полученных результатов задачи поиска можно на практике, это выполнено за нас и результаты доложены на РОМИП- Российский семинар по Оценке Методов Информационного Поиска

Эффективность описываемых алгоритмов была ндоказана а практике. Было создано 3 дорожки для сравнения алгоритмов работы поисковых систем:

- поиск по веб-коллекции (web-adhoc)
- поиск по нормативным документам (legal-adhoc)

– поиск по смешанной коллекции (mixed-adhoc)

1.3.7 Ранжирование в полнофункциональной поисковой системе

До сих пор мы уделяли внимание поиску лучших документов по запросу. Теперь рассмотрим схемы, позволяющие создать список документов, которые, вероятно, относятся к лучшим документам, соответствующим запросу.

Это должно резко снизить стоимость поиска таких документов без существенной потери релевантности результатов с точки зрения пользователя.

Следовательно, в большинстве приложений достаточно найти К документов, релевантность которых мало отличаются от наилучших.

Далее мы детализируем схемы поиска таких документов и покажем, что они позволяют избежать ранжирования большей части документов из коллекции.

Очевидно, что для запроса, состоящего из многих терминов, достаточно рассмотреть документы, содержащие по крайней мере один из этих терминов. Мы можем использовать это обстоятельство, дополнив его эвристическими правилами.

Рассмотрим только документы, содержащие термины, значение idf которых превосходит заданный порог. Таким образом, при обходе словопозиций достаточно просмотреть только термины с высоким значением idf. Это дает значительный выигрыш: инвертированные списки для терминов с низкими значениями idf, как правило, очень длинные; исключив их из рассмотрения, можно значительно сократить список документов, для которых вычисляется косинусная мера сходства. Этот эвристический прием можно интерпретировать так: термины с низким значением idf рассматриваются как стоп-слова и не вносят вклад в релевантность документа.

Например, для запроса catcher in the rye достаточно обойти только словопозиции терминов catcher и rye . Разумеется, порог отсечения можно сделать зависимым от запроса.

Рассмотрим только документы, содержащие многие (в специальном случае — все) термины запроса. Это можно сделать в процессе обхода словопозиций; релевантность вычисляется только для документов, содержащих все (или многие) термины запроса.

Недостаток этой схемы заключается в том, что, требуя, чтобы в документе были представлены все (или даже многие) термины запроса до вычисления их косинусной меры сходства, мы можем получить в результате меньше документов-кандидатов.

Идея чемпионских списков (champion lists), который иногда называют также списками фаворитов (fancy lists) или списками топ-документов (top docs), заключается в предварительном определении для каждого словарного термина t набора из r документов с наибольшими весами по отношению к термину t. При этом величина r выбирается заранее. В схеме взвешивания tf-idf эти r документов имеют наибольшие значения tf по отношению к термину t. Этот набор из r документов называется чемпионским списком для термина t.

Теперь, имея запрос, мы можем создать множество A следующим образом: объединим списки чемпионов для каждого термина, содержащегося в запросе q. Вычисление косинусной меры сходства теперь ограничивается только документами множества A.

Критический параметр к этой схеме — значение r, сильно зависящее от приложения. Интуитивно ясно, что величина r должна быть большой по сравнению с K, особенно если используется любая форма сокращения индекса, описанная выше.

Одна из проблем заключается в том, что величина r определяется в момент построения индекса, в то время как число K зависит от приложения и может оставаться неопределенным вплоть до момента, пока не будет получен запрос q. В результате мы можем (как и в случае сокращения индекса) обнаружить, что множество A содержит меньше K документов. Нет никаких причин полагать число r одинаковым для всех терминов в словаре; например, для более редких терминов его можно установить на более высоком уровне.

Во всех инвертированных списках, описанных ранее, документы были упорядочены в соответствии с некоторым общим критерием: как правило, по идентификаторам документов. Такое упорядочение позволяет осуществлять параллельный обход инвертированных списков для всех терминов запросов, ранжируя каждого обнаруженного документа. Этот процесс иногда называют ранжированием **документ за документом** (document-at-a-time scoring).

Теперь опишем метод **неточного поиска** наилучших документов, в котором не все словопозиций упорядочены одинаково, что препятствует осуществлению параллельного обхода.

Следовательно, необходимы аккумуляторы для накопления релевантности термин за термином. Такая схема называется ранжированием "термин за термином" (term-at-a-time scoring).

Идея заключается в том, чтобы документы d в инвертированном списке для термина t были расположены в порядке убывания значения tf.

Следовательно, упорядочение документов изменяется от одного инвертированного списка к другому, и мы не можем ранжировать их в ходе параллельного обхода инвертированных списков для всех терминов запроса. Если инвертированные списки составлены в порядке убывания значения tf, то существуют два способа значительно снизить количество документов , для которых аккумулируется релевантность:

1) при обходе инвертированного списка для термина запроса t просматриваем только начальный участок списка либо после того, как будет просмотрено фиксированное количество документов r, либо после того, как значение tf опустится ниже заданного порога;
2) аккумулируя релевантность во внешнем цикле алгоритма, мы рассматриваем термины запросов в порядке убывания значения idf, так что термины запроса, которые могут внести больший вклад в итоговую релевантность, рассматриваются первыми.

Последняя идея допускает адаптацию в момент обработки запроса: обнаружив термин запроса с небольшим значением idf, мы можем определить,

следует ли продолжить обработку на основе изменения релевантности документов, определенных в ходе обработки предыдущего термина запроса. Если эти изменения являются минимальными, то мы можем отказаться от аккумуляции релевантности для оставшихся терминов запроса или обрабатывать более короткие начальные участки их инвертированных списков.

Можно также реализовать вариант статического упорядочения, в котором каждый инвертированный список упорядочен на основе аддитивной комбинации статических рангов и динамических показателей, зависящих от запроса.

В этом случае согласованность упорядочения инвертированных списков также нарушается и, следовательно, термины запроса необходимо обрабатывать поочередно, накапливая релевантность для всех просмотренных документов. В зависимости от конкретных функций ранжирования инвертированный список для документа можно упорядочить в соответствии с другими показателями, отличающимися от частоты термина; эта более общая схема называется упорядочением по важности (impact ordering).

В методе **отсечения кластеров** (cluster pruning) предусмотрен предвари тельный этап, на котором происходит кластеризация векторов документов. Затем, в момент обработки запроса, мы будем рассматривать только документы, принадлежащие небольшому количеству кластеров, являющихся кандидатами для ранжирования, основанных на косинусной мере сходства. Конкретнее, предварительный (до выполнения запроса) этап выглядит так:

1) сслучайным образом извлекаем из коллекции $\sqrt{N}$ документов. Назовем их ведущими (leaders);
2) для каждого документа, не являющегося ведущим, находим ближайший к нему ведущий документ.

Затем, на этапе обработки запроса, происходит следующее:

3) По запросу q найдем ведущий документ L, ближайший к запросу q. Для этого необходимо вычислить косинусные меры сходства запроса q с каждым из ведущих документов.
4) Множество кандидатов A состоит из документа L и его ведомых документов. Основываясь на косинусной мере сходства, ранжируем все документы из множества кандидатов.

Использование случайно выбранных ведущих документов для кластеризации работает быстро и с некоторой вероятностью отражает распределение векторов документов в векторном пространстве. Область векторного пространства, плотно заполненная документами, вероятно, породит несколько лидеров, что приведет к более мелкому разбиению на подобласти (рис. 3.8).

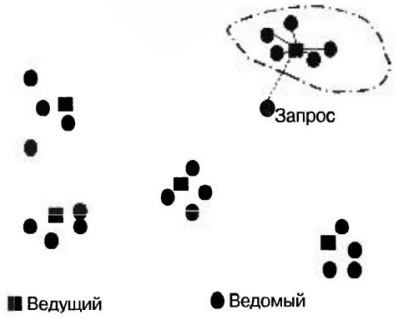


Рисунок 3.8 – Отсечение кластеров

Введя дополнительные параметры b_1 и b_2, являющиеся положительными целыми числами, можно получить разные варианты отсечения кластеров. На этапе предварительной обработки мы свяжем каждый ведомый документ с его b_1 ближайшими ведущими документами, а не просто с единственным ближайшим ведущим документом.

1.3.8 Вопросы к лекции №3

1 Как операторы языка запросов влияют на ранжирование в векторном пространстве?
2 Опишите модель векторного пространства для XML-поиска.
3 Поясните понятие параметрические и зонные индексы. Приведите примеры.
4 Поясните понятие – частота термина и взвешивание. Приведите примеры.
5 Поясните понятие – модель векторного пространства для ранжирования.
6 Поясните понятие –функция tf-idf. Приведите примеры.
7 Как выбрать из коллекции документы релевантные запросу и только их. и, возможно, упорядочить выборку по релевантности?
8 Каков состав основных этапов построения инвертированного индекса
9 В чем суть метода отсечения кластеров?
10 Дайте определение понятия релевантность.
11 Дайте определение понятия ранжирование ссылок.
12 . В чем суть процесса ранжирования *документ за документом* (document-at-a-time scoring).
13 Опишите метод *неточного поиска* наилучших документов

1.4 Лекция№ 4 Оценка информационного поиска

План лекции

1. XML-поиск
2. Факторы, влияющие на результат информационного поиска
3. Основные параметры оценки поисковой системы
4. Стандартные текстовые коллекции;
5. Оценка неранжированных результатов поиска;
6. Оценка ранжированных результатов поиска;
7. Обратная связь по релевантности и расширение запроса;
8. Качество системы и её полезность для пользователя;
9. Снипеты.

1.4.1 XML-поиск

Информационно-поисковые системы часто противопоставляются реляционным базам данных. Традиционно поисковые системы извлекали информацию из неструктурированного набора данных, под которым понимается "простой" текст без разметки. В противоположность им базы данных предназначены для обработки запросов к реляционным данным, представляющим собой совокупность записей, хранящих значения заранее определенных атрибутов, таких как табельный номер сотрудника, должность и зарплата. Между информационно-поисковыми системами и базами данных существуют фундаментальные различия в модели поиска, структурах данных и языках запроса.

Наиболее популярным из имеющихся стандартов кодирования структурированных документов считается **extensible markup language**, или язык XML. В рамках изучения дисциплины XML интересует нас лишь как язык для кодирования текстов и документов.

XML-документ (XML document) — это упорядоченное и размеченное дерево. Каждый его узел — это XML-элемент (XML element).
Он записывается с помощью открывающего и закрывающего тегов (tag). Элемент может иметь один или несколько **XML-атрибутов** (XML attributes). Пример XML документа и его преобразование в DOM-объект представлен на рисунках 4.1 и 4.2 соответственно.

```
<play>
<author>Shakespeare</author>
<title>Macbeth</title>
<act number="I">
<scene number="vii">
<title>Macbeth's castle</title>
<verse>Will I with wine and wassail     </verse>
</scene>
</act>
</play>
```

Рисунок 4.1 – Пример XML-документа

Заметим, что у этого языкового средства есть и другие точки приложения. Рассмотрим методы на основе инвертированных индексов для языка XML, ориентированного на текст.

Работа с XML-документам и их обработки основаны на DOM Document Object Model) объектной модели документа. Она имеет древовидную структуру и содержит элементы, атрибуты и текст внутри элементов как узлы дерева. Рис. 4.2 представляет собой упрощенное представление документа, изображенного на рис. 4.1, с помощью модели DOM.

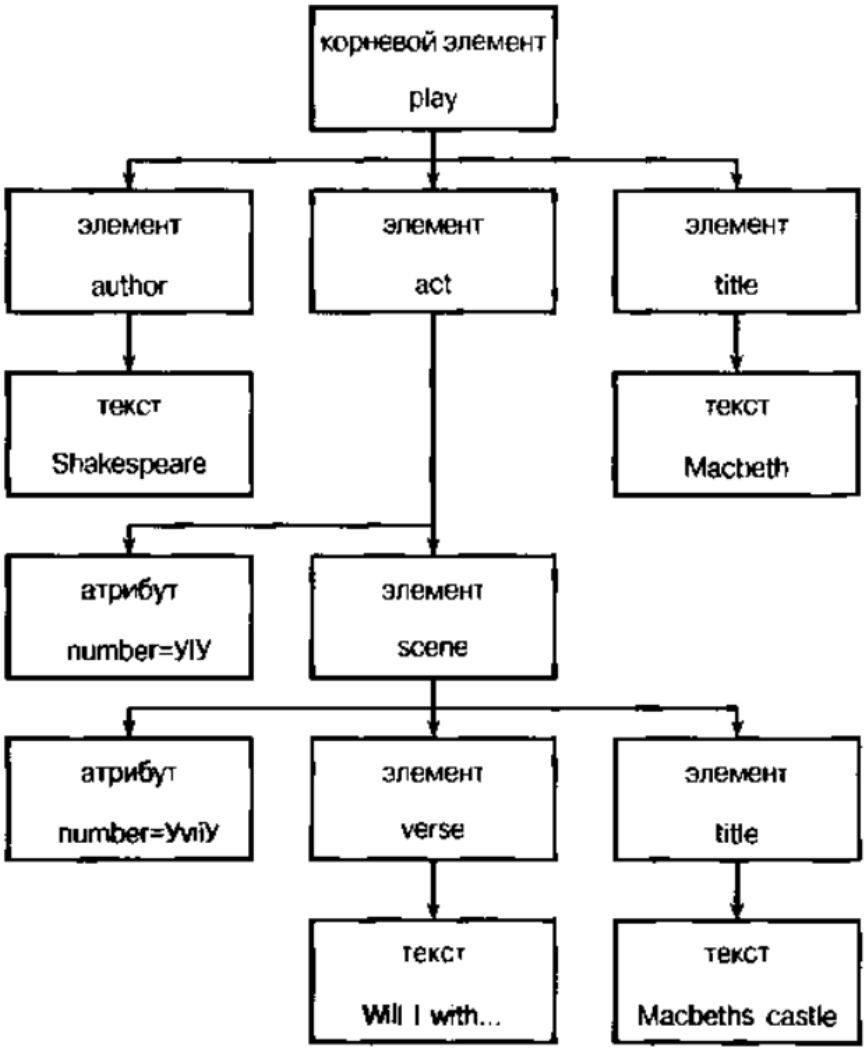


Рисунок 4.2- – Преобразование XML в DOM-объект

С помощью интерфейса прикладного программирования модели DOM можно обработать XML-документ, начиная с корневого узла и спускаясь вниз по дереву от родительских узлов к дочерним.

Для перебора путей в коллекции XML-документов обычно используется язык XPath. Эти пути называются контекстами или контекстами XML(XML contexts). Для целей поиска и обработки текстов можно ограничиться лишь

небольшим подмножеством языка XPath. Выражение языка XPath node выбирает все узлы по заданному имени.

Схема накладывает определенные ограничения на структуру XML-документов, допустимых в конкретном приложении.. Существует два стандарта схем для XML-документов: XML DTD (document type definition) и XML (XML schema). Пользователи могут задавать структурированные запросы системе поиска по XML-документам, только если у них есть хотя бы минимальные знания о схеме коллекции.

1.4.2 Факторы, влияющие на результат информационного поиска

Информационный поиска состоит из нескольких взаимосвязанных этапов, их состав представлен на рисунке 4.3:

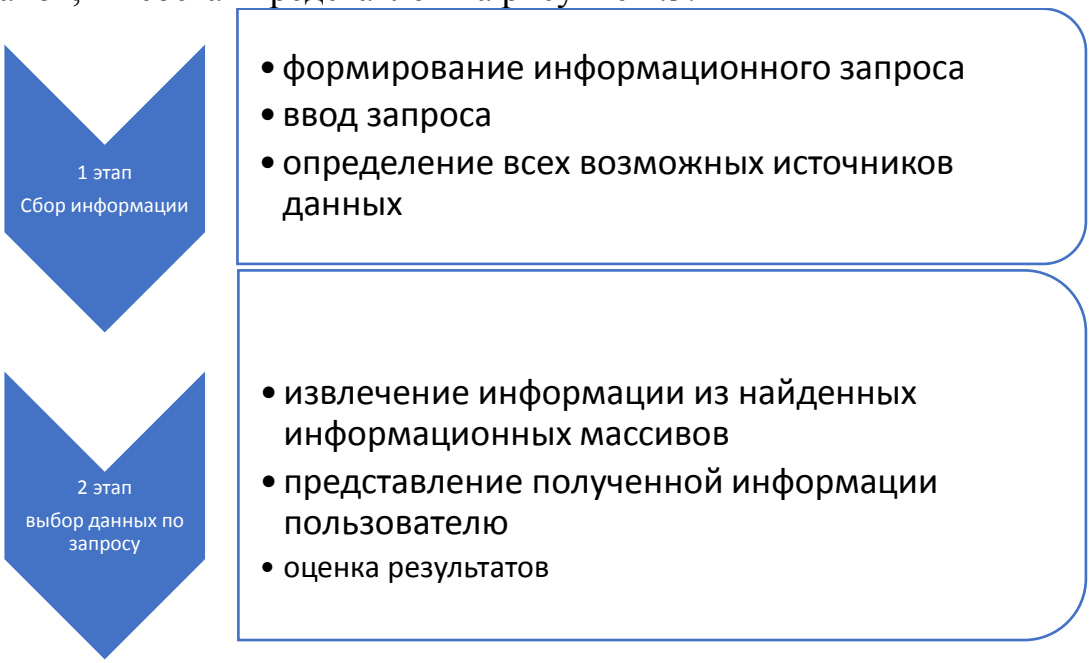


Рисунок 4.3- – Состав этапов информационного поиска

Результаты поиска характеризуются степенью соответствия найденных данных информационному запросу - релевантностью.

При этом правильная формулировка запроса очеbнь важна. В современных поисковых системах имеются инструменты уточнения запроса. Естественно на отображение выборки влияет выбор поисковой системы.

Поисковая система состоит из двух частей:

1. программа, сбора информации, индексации,
2. программный комплекс выбирающий данные по запросу пользователя и ранжирующий для представления результата.

Паук (поисковый робот) – **программа, осуществляющая поиск источников данных.** Паук свободно перемещается по всемирной паутине, попадая на одну страницу, он ищет на ней ссылки на другие страницы и заходит на каждую из них, повторяя предыдущие действия. Параллельно практически паук индексирует, собирая основные сведения о сайте в базе данных, делает копию каждой соответствующей параметрам поиска страницы и отправляет в архив. Страницы проверяются на вирусы, технические ошибки и

плагиат. Не прошедшие «кастинг» страницы отсеиваются до попадания в индекс, ясно, что сайт тем быстрее попадет в индекс чем больше ссылок на него имеется.

Кроме качественных проверок действуют количественные ограничения размещения в индекс. Это объясняется ограниченностью ресурсов поисковой системы. В этой связи каждый сайт имеет "предел" - количество страниц, которые паук может обойти за один раз, и максимум проиндексированных документов. На крупных сайтах для направления работы "паука" располагают специально разработанные карты сайта.

Одной из важных задач поиска является сокращение сроков индексации. Ресурсов в интернете становится больше с каждой секундой, поэтому был создан быстроробот.

Быстроробот – это программа, занимающаяся индексацией часто обновляемых сайтов (блогов, новостных порталов, соц. сетей и т. д.). Эта программа позволяет разместить на первые места самые свежие результаты поиска. Быстроробот не учитывает ссылки, то есть одна и та же страница может быть проиндексирована и роботом и быстроробото́м.

Механизм действия программного комплекс выбора данных по запросу рассмотрим на примере Яндекса [16].

Введенный пользователем запрос вначале обрабатывает **балансировщик нагрузки**, который автоматически распределяет запросы по некоторым объединениям серверов (кластерам). Это необходимо для эффективного использования вычислительной мощности.

Затем выполняется проверка наличия готовых результатов в кэше.

Следующая операция - метапоиск выявляет тип данных поискового запроса. Обычно это текст и могут быть изображения. Одновременно практически проверяется текст на орфографию, выясняется география отправки запроса. Кроме того, выявляется коммерческая направленность запроса.

Специфическими считаются запросы, содержащие “скачать”, “купить” и др. Для разных видов запросов подходят различные факторы ранжирования если для новостей важна свежесть информации, то для других название бренда.

Система собирает информацию для нового списка по запросу в "базовый поиск". В нем хранится индекс поисковой системы, разделенный на несколько частей и обрабатываемый на разных серверах. Наличие нескольких серверов позволяет снизить нагрузку и ускорить процесс поиска. При этом, для защиты от потери информации, каждый сервер создает несколько копий.

Базовый поиск возвращает метапоиску найденные результаты. Хотя этот ответ еще не является конечным. В Яндексе результаты обрабатываются сначала фильтрами, а потом готовятся к выдаче алгоритмом Матрикснет [16].

Фильтр[16]. – это алгоритм, содержащий определенные требования к сайтам. Несоответствующие этим параметрам исключаются из индекса.

Матрикснет [16].– это алгоритм машинного обучения, который строит формулу ранжирования результатов поисковой выдачи

После появления этого алгоритма релевантность выдачи Яндекса значительно возросла, и пользователь получает необходимый ответ

практически мгновенно. Но **скорость** не самый важный критерий для пользователя.

1.4.3 Критерии оценки качества информационно-поисковой системы.

Различают два типа оценок:

1. оценки-описания;
2. оценки-шкалы,

Значения оценки-описания характеризуют непосредственно систем и позволяют достаточно полно судить о существенных свойствах оцениваемых объектов. Например, предсказывать их поведение в тех или иных конкретных условиях. В этом случае «оценка-описание» называется эффективной.

Значения оценки-шкалы определяют сравнительные достоинства различных поисковых и упорядочивают множество оцениваемых объектов. Например, различных ИПС, не вступая при этом в противоречие с существующими у нас содержательными представлениями о сравнительных достоинствах этих объектов. В этом случае «оценка-шкала» называется здравой.

Содержательные представления о сравнительных достоинствах систем являются содержательной оценкой. Объективная формальная оценка не должна противоречить содержательной.

Оценки поисковых систем делят на два класса, которые называются **внешними** (или функциональными) и **внутренними оценками.**

Внешние, или функциональные, оценки основаны на сравнении результатов работы системы с результатами идеального содержательного поиска, осуществляемого экспертом. В теории информационного поиска для этого введены понятия релевантности и пертинентнести.

Под **релевантностью** понимается соответствие выдачи запросу, т.е. релевантность характеризует качество алгоритма поиска. Под пертинентностъю - соответствие выдачи потребностям лица (или лиц), для которого (которых) осуществляется поиск информации, т. е. пертинентность характеризует смысловыражающие возможности ИПЯ, точность отображения с его помощью информационных потребностей.

В настоящее время иногда термин релевантность используют в более широком смысле, и различают релевантность первого рода (формальную релевантность), которая соответствует термину, первоначально введенному в теории информационного поиска и релевантность второго рода, соответствующую понятию пертинентности.

Основные критерии оценки поисковиков

Состав комплекс критериев оценки качества информационного поиска зависят от конкретного назначения и принципов реализации поисковой системы. К таким критериям относятся:

— Точность выдачи - отношение числа выданных релевантных документов к сумме числа выданных релевантных и числа выданных нерелевантных документов.

— Полнота выдачи - отношение числа выданных релевантных документов к сумме числа выданных релевантных и числа не выданных релевантных документов.

— Потери информации - отношение числа не выданных релевантных документов к сумме числа выданных релевантных и числа не выданных релевантных документов.

— Информационный шум - отношение числа выданных нерелевантных документов к сумме числа выданных релевантных и числа выданных нерелевантных документов.

— Чувствительность - отношение числа выданных релевантных документов к сумме числа выданных релевантных и числа не выданных релевантных документов.

— Специфичность - отношение числа невыданных нерелевантных документов к сумме числа выданных нерелевантных и числа невыданных нерелевантныхдокументов.

На практике для сравнения поисковых систем используются усредненные графики зависимости полноты от точности. Чтобы избежать сравнения пар полнота - точность используются однозначные оценки. Одной из таких оценок является E-мера, позволяющая избежать сравнения пар полнота - точность за счет введения отношения их значимости.

$$E(b) = 1 - \frac{(b^2+1.0)PR}{b^2P+R} \qquad (4.1)$$

где P - точность,
R - полнота,
b - отношение значимости полноты и точности.

1.4.4 Стандартные текстовые коллекции

Для оценки стандартным способом информационно-поисковой системы по произвольным запросам необходима тестовая коллекция, состоящая из трех компонентов:

— коллекция документов;

— набор тестовых информационных потребностей, выраженных в виде запросов;

— набор оценок релевантности представленных, как правило, в виде бинарных утверждений релевантности и нерелевантности (true и false).

Стандартный подход к оценке информационно-поисковых систем основан на понятии релевантных документов. Документ из тестовой коллекции релевантный или нерелевантный. Эта градация называется эталонной оценкой релевантности (gold I standard or ground truth). Коллекция тестовых документов и набор информационных потребностей должны иметь достаточный объем.

Оценку усредняют по всей совокупности тестов. В качестве первого очень грубого приближения достаточным минимумом считается набор из 50 информационных потребностей.

Документ является релевантным, если он соответствует заданной информационной потребности, а не просто если он содержит все слова из запроса. Для того чтобы оценить систему, необходимо явно сформулировать информационную потребность, относительно которой можно судить о релевантности или нерелевантности найденных документов. Для простоты можно допустить, что релевантность можно оценить по шкале, т.е. одни документы являются сильно релевантными, а другие — слабо.

1.4.5 Оценка неранжированных наборов результата поиска

Точность (P): доля релевантных документов из всех найденных.

P(релевантные|найденные) == #(найденные релевантные объекты) / #(найденные объекты)

Позволяет определить "степень надежности" системы. Не учитывает общее количество документов.

Полнота (R): доля найденных релевантных документов из всех релевантных в коллекции.

Позволяет определить "степень полноты" системы.

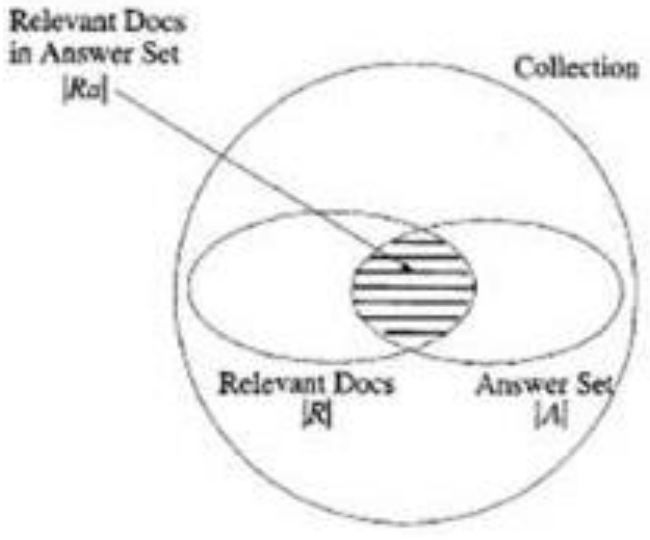


Рисунок 4.4. Графическое представление обрабатываемой коллекции документов.

Точность не совсем подходит как характеристика ИП. Бывает в вебе много искажённых данных. Если система, настроена на максимизацию точности она будет почти каждый документ объявлять нерелевантным.

Точность для информационного поиска скорее - доля правильных классификаций.

$$P = \frac{TP}{retrieved} = \frac{TP}{TP+FP} \quad (4.2)$$

$$R = \frac{TP}{relevant} = \frac{TP}{TP+FN} \quad (4.3)$$

Точность - доля правильных классификаций.

$$accuracy = \frac{TP+TN}{TP+TN+FP+FN}$$

Можно получать высокую полноту (но низкую точность), извлекая все документы для всех запросов. Полнота является неубывающей функцией от количества найденных документов. Точность обычно падает (в хороших системах). Точность может быть вычислена на разных уровнях полноты. Пользователи, ориентированные на высокую точность - веб-серферы, на высокую полноту - профессиональные исследователи, юристы, аналитики.

F-мера является комбинированной мерой, оценивающей компромисс между точностью и полнотой (взвешенное среднее гармоническое):

$$F = \frac{1}{\alpha\frac{1}{P}+(1-\alpha)\frac{1}{R}} = \frac{(\beta^2+1)PR}{\beta^2 P+R} \quad \beta^2 = \frac{1-\alpha}{\alpha} \quad (4.4)$$

При значении в<1 акцент делается на точности, при в>1 - на полноте.

Обычно используется сбалансированная F-мера, т.е. в=1 или б=S

$$F_{\beta=1}\ \frac{2PR}{P+R} \quad (4.5)$$

Когда значения двух чисел отличаются, среднее гармоническое ближе к их минимуму, чем среднее арифметическое или геометрическое. Например, если 1 из 10000 документов релевантен, мы можем получать 100% полноты, извлекая все документы. Среднее арифметическое будет 50%, а гармоническое - 0,02%.

Полнота, точность и F-мера являются мерами, основанными на множествах (например, неупорядоченный набор документов). В ранжированных поисковых системах значения P и R связаны с позицией в рейтинге. Оценка производится путем вычисления точности, как функции от полноты. Если (k+1)-ый найденный документ релевантен, то R(k+1) > R(k), а P(k+1) > P(k). Если (k+1)-ый найденный документ нерелевантен, то R(k+1) = R(k), но P(k+1) < P(k). Чтобы удалить колебания, используется интерполированная точность.

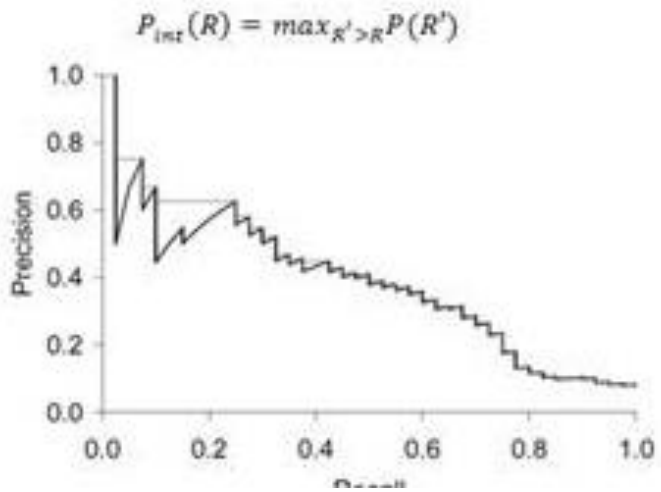


Рисунок 4.4. Графическое представление зависимости точности от полноты.

Одиннадцатиточечная интерполированная средняя точность (11-point interpolated average precision). Измеряется точность на 11 уровнях полноты {0.0, 0.1, 0.2,…,1.0}, затем рассчитывается среднее арифметическое уровня точности.

Чтобы найти среднее значение средней точности (mean average precision (MAP)), вычисляется средняя точность (AP) для каждого информационного запроса. Затем значение средней точности получается для набора из первых k документов, имеющихся после каждого нахождения релевантного документа. MAP = среднее значение AP множества информационных потребностей.

$$\mathrm{MAP(Q)} = \frac{1}{|Q|}\sum_{j=1}^{|Q|}\frac{1}{m_j}\sum_{k=1}^{m_j}\mathrm{P}(\Re_k) \quad q_j \in Q \tag{4.6}$$

Запрос?

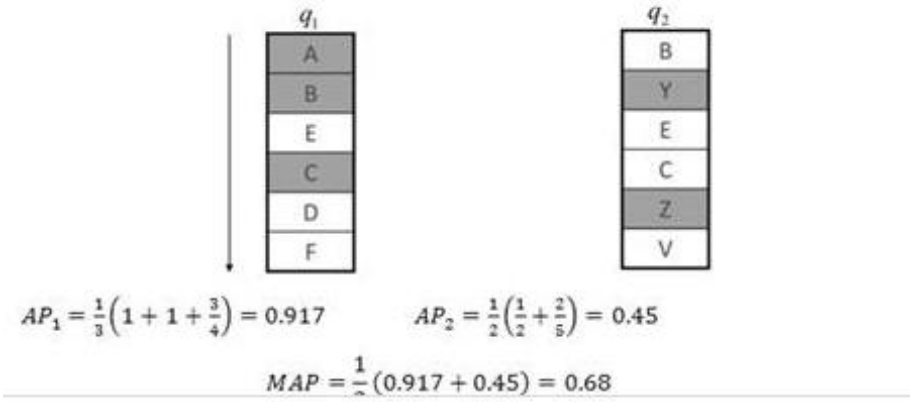
$\{d_1, \ldots, d_{m_j}\}$

- документы, релевантные запросу

q_j, $\Re_k$

Ранжированный набор первых k найденных результатов.

Для одной информационной потребности AP связана с площадью под неинтерполированной кривой точности/полноты. Пример:



Расчет точности для k. Фиксируется k - количество извлекаемых результатов, например k=10. Вычисляется точность для первых k объектов. Достоинство: не требуется множества релевантных документов (полезно для веб-поиска). Недостаток: общее количество релевантных документов сильно влияет на точность для k. Например, если количество релевантных документов равно 8, то точность для 20 будет не более 0,4.

R-точность. Для известного релевантного множества размера Rel вычисляется количество релевантных документов r среди первых Rel результатов поиска. Достоинство: идеальная система достигает R-точность = 1,0. Недостаток: рассматривается только одна точка на кривой точность/полнота.

Операционная характеристика приемника (receiver operating characteristic). ROC-кривая отображает график зависимости доли верно положительных классификаций (чувствительности) от доли ложно положительных классификаций (1 - специфичность).

Доля TP = чувствительность = полнота = TP / (TP + FN)

Доля FP = 1 - специфичность = FP / (FP + TN)

Человек — это не робот, невозмутимо возвращающий стандартные выводы о релевантности документа по отношению к запросу. Его суждения о релевантности носят субъективный характер. Тем не менее успех информационно-поисковой системы зависит от того, как она удовлетворяет информационные потребности пользователей.

1.4.6 Оценка ранжированных результатов поиска

Определение ранжирования и релевантности рассмотрены в лекции№3

Рассмотрим N объектов $U=\{u_i\}^N_{i=1}$ и M элементов $E=\{e_j\}^M_{j=1}$.

Результат работы алгоритма ранжирования элементов E для объекта $u \in U$— это отображение $r: E \to R$, которое сопоставляет каждому элементу $e \in E$

вес *r(e)*, характеризующей степень релевантности элемента объекту (чем больше вес, тем релевантнее объект)[13]. При этом, набор весов $\{r(e)\}_{e \in E}$ задает перестановку $\pi:[1...M] \to [1...M]$ на наборе элементов элементов *E* (считаем, что множество элементов упорядоченное) исходя из их сортировки по убыванию веса *r(e)*.

Чтобы оценить качество ранжирования, необходимо иметь некоторый «эталон», с которым можно было бы сравнить результаты алгоритма. Рассмотрим $r^{true}:E \to [0,1]$— эталонную функцию релевантности, характеризующую «настоящую» релевантность элементов для данного объекта ($r^{true}=1$ — элемент идеально подходит, $r^{true}=0$ — полностью нерелевантен), а так же соответствующую ей перестановку $\pi^{true}:E \to [1...M]$ (по убыванию $r^{true}(e)$).

Существует два основных способа получения r^{true}:

1. На основе исторических данных. Например, в случае рекомендаций контента, можно взять просмотры (лайки, покупки) пользователя и присвоить просмотренным весам соответствующих элементов 1 ($r^{true}(e) \leftarrow 1$), а всем остальным — 0.
2. На основе экспертной оценки. Например, в задаче поиска, для каждого запроса можно привлечь команду асессоров, которые вручную оценят релевантности документов запросу.

Стоит отметить, что когда r^{true} принимает только экстремальные значения: 0 и 1, то престановку π^{true} обычно не рассматривают и учитывают лишь множество релевантных элементов, для которых $r^{true}=1$.

Цель метрики качества ранжирования — определить, насколько полученные алгоритмом оценки релевантности *r(e)* и соответствующая им перестановка π соответствуют *истинным* значениям релевантности r^{tru}. Рассмотрим основные метрики.

Mean average precision

Mean average precision at K (map@K) — одна из наиболее часто используемых метрик качества ранжирования. Чтобы разобраться в том, как она работает начнем с «основ».

*Замечание: "*precision" метрики используется в бинарных задачах, где r^{tru} принимает только два значения: 0 и 1.*

Precision at K

Precision at K (p@K) — точность на K элементах — базовая метрика качества ранжирования для одного объекта. Допустим, наш алгоритм ранжирования выдал оценки релевантности для каждого элемента $\{r(e)\}_{e \in E}$. Отобрав среди них первые элементов с наибольшим *r(e)* можно посчитать долю релевантных. Именно это и делает precision at K:

$$p@K = \frac{\sum_{k=1}^{K} r^{true}(\pi^{-1}(k))}{K} = \frac{\text{релевантных элементов}}{K}.$$

Замечание: под $\pi^{-1}(k)$ понимается элемент $e \in E$, который в результате перестановки оказался на -ой позиции. Так, $\pi^{-1}(1)$ — элемент с наибольшим r(e , $\pi^{-1}(2)$ — элемент со вторым по величине r(e) и так далее.

Average precision at K

Precision at K — метрика простая для понимания и реализации, но имеет важный недостаток — она не учитывает порядок элементов в «топе». Так, если из десяти элементов мы угадали только один, то не важно на каком месте он был: на первом, или на последнем, — в любом случае p@10=0,1. При этом очевидно, что первый вариант горазо лучше.

Этот недостаток нивелирует метрика ранжирования **average precision at K (ap@K)**, которая равна сумме p@k по индексам k от 1 до K *только для релевантных элементов*, деленому на K:

$$ap@K = \frac{1}{K}\sum_{k=1}^{K} r^{true}(\pi^{-1}(k)) \cdot p@k.$$

Так, если из трех элементов мы релевантным оказался только находящийся на последнем месте, то $ap@3 = \frac{1}{3}(0 + 0 + 1/3) \approx 0.11$, если угадали лишь тот, что был на первом месте, то $ap@3 = \frac{1}{3}(1/1 + 0 + 0) \approx 0.33$, а если угаданы были все, $ap@3 = \frac{1}{3}(1/1 + 2/2 + 3/3) = 1$.

1.4.7 Сниппеты

Сниппет (в переводе с английского snippet — отрывок, часть) — это фрагмент текста страницы сайта, наиболее релевантный запросу пользователя, в выдаче поисковой системы размещается под ссылкой на ресурс. Он сообщает посетителю, какую информацию тот получит, перейдя по данному линку.

Сниппеты состоят из заголовка, ссылки, текста описания, URL страницы, ее размера и других данных в зависимости от используемой поисковой системы. Например, в Яндекс сниппете дополнительно содержится последняя копия найденной страницы и наименование раздела Яндекс-каталога, в котором она зарегистрирована. Google может выборочно показывать рубрики сайта или предлагать перевести текст страницы с помощью Google переводчика.

Роль сниппета в раскрутке сайта

В зависимости от вида и информативности сниппета пользователь принимает решение о необходимости просмотра найденной страницы. Сайт, показываемый, к примеру, на седьмой позиции в поисковой выдаче может иметь больше целевого трафика, чем ресурс из ТОП-3, благодаря привлекательности сниппета.

Составление сниппетов

В процессе поисковой оптимизации сайта специалист по продвижению не может однозначно задать значение сниппета. Но существует ряд мер, оказывающих влияние на выбор поисковиком фрагмента для показа в выдаче.

Title. Обычно включается в состав сниппета. Данный тег прописывается для каждой страницы отдельно, в него вносится основной продвигаемый запрос. Текст в title должен кратко описывать содержание страницы. Например, «определение и классификация современных молодежных субкультур» вместо «молодежные субкультуры». Размер тега не должен превышать 70 символов с пробелами: таково максимальное число знаков, отображаемое поисковой системой Google. Если ориентироваться на другие поисковики (для Rambler пограничное значение — 118 символов), часть информации может оказаться утерянной.

Description. В сниппет может включаться часть тега description, если в нем обнаружен запрос, введенный пользователем. Максимальная длина его отображения в сниппете составляет 200 знаков для Яндекса и 156 для Google.

Контент. В сниппете может показываться часть текста страницы с ключевыми словами в наиболее релевантных пассажах — достаточно информативных, длинных предложениях. Если в статье такой фрагмент будет в единичном экземпляре, именно его поисковая система выберет для формирования аннотации.

При раскрутке сайта с первого раза сложно получить привлекательные сниппеты в выдаче. На основе анализа сформированных различными поисковыми машинами результатов вносят корректировки в оптимизацию страниц: неудачные фразы исключают из индексации или заменяют в них ключи синонимами, исправляют теги и т.д.

1.4.8 Вопросы к лекции №4

1 Назовите и опишите основные факторы, влияющие на результат информационного поиска
2 Назовите и опишите основные способы оценки информационного поиска.
3 В чем суть идеи обратной связи по релевантности?
4 Каковы основные параметры оценки поисковой системы
5 Расскажите о стандартных текстовых коллекциях, их организации;
6 Каким образом производится оценка неранжированных результатов поиска;
7 Каким образом производится оценка ранжированных результатов поиска;
8 Назовите и опишите глобальные методы переформулирования запроса.
9 Назовите методы XML-поиска, ориентированные на текст и на данные.

2 Обработка естественного языка

2.1 Лекция № 5 Введение в обработку естественно- языковых текстов.

План лекции№5

Общее понятие процесса обработки естественно- языковых текстов
Методика обработки естественно- языковых текстов и машинное обучение
Плоская кластеризация:
Иерархическая кластеризация
Разложение матриц и латентно-семантическое индексирование
Основы поиска в вебе
Обход и индексирование веба

2.1.1 Общее понятие процессов обработки естественно- языковых текстов

Обработка естественного языка (NLP) — это совокупность компьютерных наук, в том числе искусственного интеллекта (ИИ), которое занимается предоставлением компьютерам возможности понимать и обрабатывать человеческий язык.

Основная задача НЛП —программирование компьютеров для анализа и обработки огромного количества данных на естественном языке.

Классификация текстов

Понятие классификации является очень общим и имеет много приложений как в области информационного поиска, так и за его пределами.

В задаче классификации текстов задано описание документа $d \in X$, где X — пространство документов, и фиксированное множество классов $C = \{c_1, c_2, \dots c_j\}$. Классы также называются категориями и метками. Как правило, пространство документов X имеет большую размерность, а классы определяются экспертами в зависимости от приложения. Кроме того, задано обучающее множество B размеченных документов $\langle d,c \rangle$, где $\langle d,c \rangle \in X \times C$. Используя метод обучения, или алгоритм обучения, мы хотим получить классификатор, или функцию классификации y, отображающую документы в классы. Этот метод называется обучением с учителем (человек, который определяет классы и готовит обучающие документы) играет роль учителя, управляющего процессом обучения. Обозначим метод обучения с учителем буквой Г и запишем $Г(B) = y$. Метод обучения Г получает на вход обучающее множество B и возвращает функцию классификации y.

Классы в задачах классификации текстов часто имеют определенную структуру, например иерархическую. В ней существует по две категории для региона, отрасли промышленности и тематической области. Иерархия может играть важную роль в решении задачи классификации.

Метод обучения с учителем, который мы рассмотрим, — мультиномиальный наивный метод Байеса. В этом методе вероятность того, что документ d принадлежит классу c, вычисляется следующим образом.

$$P(c|d) = P(c)\prod P(t|c) \qquad (5.1)$$

Здесь P(t|c) — условная вероятность того, что термин t появится в документе из класса c. P(t|c) — это наша оценка вклада термина t в то, что документ принадлежит классу c, а P(c) — априорная вероятность того, что документ принадлежит классу c. Если термины документа не позволяют четко отделить один класс от другого, то следует выбрать тот из них, который имеет более высокую априорную вероятность. Последовательность $\{t_1, t_2, \ldots, t_n\}$ состоит из лексем документа d, являющихся частью лексикона, используемого для классификации, а n_d — количество таких лексем в документе d. Цель классификации текстов — найти наилучший класс для документа. В методе NB наилучшим считается наиболее вероятный класс, или класс c_{map}, имеющий максимальную апостериорную вероятность.

Выбор признаков — это выбор подмножества терминов, встречающихся в обучающем множестве, и использование только этого набора признаков при классификации текстов. Выбор признаков преследует две основные цели. Во-первых, это повышает эффективность обучения и применения классификатора за счет уменьшения размера лексикона. Данное обстоятельство особенно важно для классификаторов, обучение которых, в отличие от наивного байесовского метода, очень затратно. Во-вторых, выбор признаков часто повышает точность классификации благодаря исключению шумовых признаков. **Шумовой признак** — это признак, при добавлении которого к представлению документа ошибка классификации на новых данных возрастает.

2.1.2Классификация в векторном пространстве

На рис. 5.1 показаны три класса на двумерной плоскости: China, UK и Kenya.

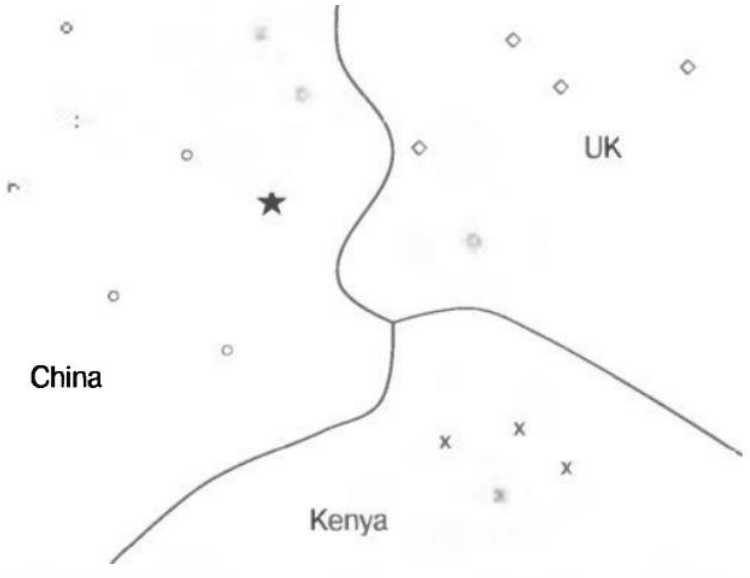


Рисунок 5.1 – Классификация на классы в векторном пространстве

Документы отмечены кружочками, ромбиками и крестиками. Разделяющие границы на рисунке выбраны таким образом, чтобы разделить три класса, но в остальном их свойства произвольны. Для классификации нового документа, отмеченного на рисунке звездочкой, мы определяем область, в которую он попал, а затем класс, соответствующий этой области (в данном случае это класс China). Векторная классификация сводится к разработке алгоритма, вычисляющего «хорошие» границы, где термин «хорошие» означает высокую точность классификации на данных, не использованных в ходе обучения.

Граница между двумя классами в методе Роккио представляет собой множество точек, равноудаленных от двух центроидов. Например, $|a_1| = |a_2| > |b_1| = |b_2|$ и $|c_1| = |c_2|$. Это множество точек всегда образует линию. Обобщением этой линии в M-мерном про странстве является гиперплоскость, которая представляет собой множество точек x.

Таким образом, границами областей классов в методе Роккио являются гиперплоскости. Правило классификации в алгоритме Роккио заключается в определении области, в которую попадает точка. Это эквивалентно поиску центроида $\vec{\mu}(c)$, к которому точка лежит ближе, чем к другим центроидам, и приписыванию этой точки к классу. В качестве примера рассмотрим звездочку на рис. 5.2. Она лежит в области China, поэтому алгоритм Роккио относит ее к классу China.

Для векторной классификации документов необходимо определить границы между классами, поскольку именно они определяют результат классификации. Вероятно, наиболее известным методом определения этих границ является метод Роккио, в котором для идентификации границ используются центроиды. Центроид класса вычисляется как усредненный вектор, или центр масс членов класса.

$$\vec{\mu}(c) = \frac{1}{|D|}\sum \vec{v}(d) \qquad (5.2)$$

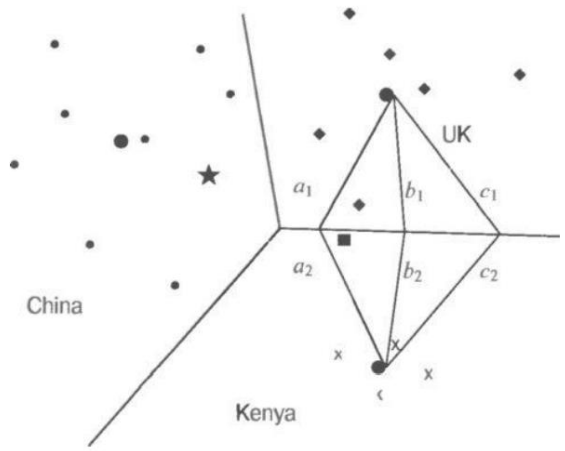


Рисунок 5.2 – Классификация Роккио

Метод 1NN не очень устойчив. Классификация каждого текстового документа зависит от класса, к которому относится отдельный обучающий документ, который может иметь неверную метку или вообще быть нетипичным. Метод KNN при K > 1 является более устойчивым. Он приписывает документы к превалирующему классу по k ближайшим соседям, случайным образом разрывая связи между ними.

Существует вероятностный вариант метода KNN. Можно оценить вероятность того, что документ принадлежит классу, как долю k-ближайших соседей в классе. На рис. 5.3 приведен пример классификации при k = 3. Оценки вероятностей того, что документ, отмеченный звездочкой, принадлежит трем классам, таковы: (класс кружочков|звездочка) = 1/3, (класс крестиков|звездочка) = 2/3, (класс ромбиков|звездочка) = 0.

Оценки метода 3NN (P_1(класс кружочков|звездочка) = 1/3) и метода 1NN (P_1(класс кружочков|звездочка) =1) отличаются, поэтому метод 3NN отдает предпочтение классу крестиков, а метод 1NN — классу кружочков.

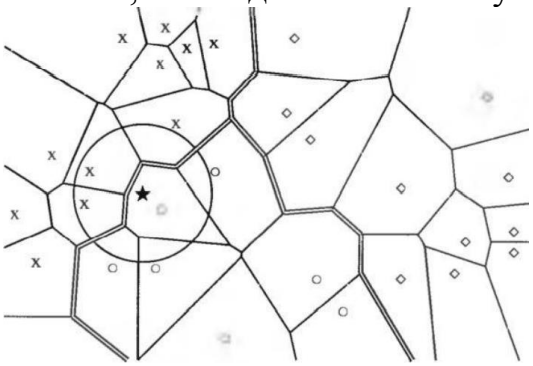


Рисунок 5.3 – Разделение в методе 1NN

Параметр K в методе KNN часто выбирается на основании опыта или знаний о решаемой задаче классификации. Желательно, чтобы параметр был нечетным, чтобы уменьшить вероятность «ничьи». Чаще всего выбираются значения к = 3 и к = 5, но используются и большие значения, между 50 и 100. В качестве альтернативы параметр можно выбрать так, чтобы он гарантировал наилучшие результаты на отложенной части обучающего множества.

2.1.3Методика обработки естественно- языковых текстов и машинное обучение

Если обучающее множество содержит два класса данных, допускающих линейное разделение, то существует большое количество линейных классификаторов, с помощью которых можно разделить эти данные. Интуитивно ясно, что разделяющая поверхность, проходящая через середину полосы, разделяющей два класса, лучше, чем разделяющая поверхность, лежащая очень близко к экземплярам одного или обоих классов. В то время как одни методы обучения, такие как персептрон, позволяют найти хотя бы один линейный разделитель, другие методы, такие как наивный байесовский метод, находят наилучший линейный разделитель, используя определенный критерий.

В частности, метод **опорных векторов** ищет разделяющую поверхность, максимально удаленную от любых точек данных. Расстояние между этой

поверхностью и ближайшей точкой данных называется зазором классификатора. В методе опорных векторов обязательно подразумевается, что решающая функция полностью определяется (обычно малым) подмножеством данных, влияющих на положение разделителя. Эти точки называются опорными векторами (в векторном пространстве точку можно рассматривать как вектор между началом координат и этой точкой). Зазор и опорные векторы для простой задачи показаны на рис. 5.4. Другие точки данных не влияют на выбор разделяющей поверхности.

Рисунок 5.4 – Опорные вектора

Максимизация зазора выглядит хорошей идеей, поскольку точки, лежащие вблизи разделяющей поверхности, порождают большую неопределенность; с вероятностью 50% классификатор может принять любое из двух решений. Классификатор с большим зазором снижает неопределенность решения. Тем самым он создает определенный запас надежности: небольшая ошибка измерения или небольшое изменение документа не приведет к неправильной классификации. Другое интуитивное обоснование метода опорных векторов продемонстрировано на рис. 5.5. По своей конструкции классификатор SVM требует, чтобы вокруг разделяющей поверхности был широкий зазор. Если попытаться поместить между классами широкую полосу, то диапазон углов, при котором это можно сделать, окажется намного меньшим, чем для гиперплоскости. В результате емкость запоминания модели уменьшается, и можно ожидать, что способность модели правильно обобщать тестовые данные увеличивается.

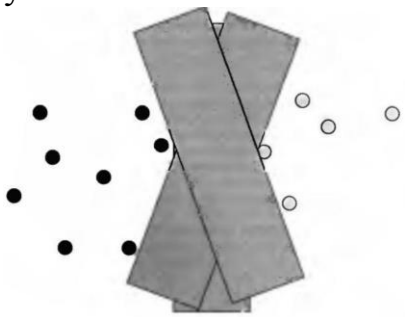


Рисунок 5.5 – Интуитивно понятное обоснование классификации с широким зазором

Приведем формальное алгебраическое описание метода опорных векторов. Разделяющая гиперплоскость задается параметром сдвига b (точкой пересечения с осью x) и вектором нормали к разделяющей гиперплоскости w. В литературе по методам машинного обучения этот вектор обычно называется вектором весов. Для того чтобы среди всех гиперплоскостей, перпендикулярных вектору нормали, выбрать одну нужную гиперплоскость, используется параметр b. Поскольку разделяющая гиперплоскость перпендикулярна вектору нормали, все точки x на гиперплоскости удовлетворяют уравнению wx = -b. Теперь допустим, что у нас есть обучающее множество D = {(x,y)}, в котором каждый элемент представляет собой пару, состоящую из точки x и соответствующей метки класса y. В методе опорных векторов два класса всегда называются +1 и -1 (а не 1 и 0), а параметр сдвига всегда явно обозначается буквой b (а не включается в вектор w в качестве постоянного слагаемого). Благодаря этому математические выкладки становятся намного яснее. В этом случае линейный классификатор описывается следующей формулой.

$$f(\vec{x}) = sign(\overrightarrow{w^T}\vec{x} + b) \quad (10)$$

Значение -1 обозначает один класс, а +1 — следующий.

2.1.4 Плоская кластеризация

Алгоритмы кластеризации разделяют совокупность документов на подмножества, или **кластеры**. Цель этих алгоритмов — создать кластеры, однородные внутри, но четко отличающиеся друг от друга. Иначе говоря, документы внутри кластера должны быть максимально похожими друг на друга, но в то же время максимально отличаться от документов другого кластера. **Кластеризация** — это наиболее распространенная форма обучения без учителя. В задачах кластеризации распределение и структура данных определяют принадлежность к кластеру.

Плоская кластеризация порождает совокупность кластеров, не имеющих явных взаимосвязей.

2.1.5 Иерархическая кластеризация

Иерархическая кластеризация создает иерархию кластеров. Различают жесткую и мягкую кластеризацию. **Жесткая кластеризация** вычисляет жесткое присваивание — каждый документ может быть элементом только одного кластера. Присваивание в алгоритмах **мягкой кластеризации** является мягким, принадлежность документа распределена по всем кластерам. При мягком присваивании документ может частично принадлежать нескольким кластерам. Примером алгоритма мягкой кластеризации является латентное семантическое индексирование — одна из форм снижения размерности.

Кластерная гипотеза — это основное предположение, позволяющее применять кластеризацию для информационного поиска. Документы, принадлежащие одному и тому же кластеру, являются примерно одинаково релевантными по отношению к информационным потребностям.

По умолчанию результаты поиска в информационно-поисковых системах представляются в виде простого списка. Пользователи просматривают этот список сверху вниз, пока не найдут нужную им информацию. В то же время после кластеризации результатов поиска похожие документы будут указаны в списке недалеко друг от друга. Иногда легче просматривать группы однородных документов, а не множество отдельных документов. Это особенно полезно, если термин запроса имеет несколько значений. Например, запрос jaguar. Три наиболее распространенных значения этого слова — автомобиль, животное и операционная система компьютера Apple.

Приложение кластеризации использует гипотезу о кластерах напрямую для улучшения результатов поиска за счет кластеризации всей коллекции. Для идентификации начального множества документов, соответствующих запросу, используется стандартный инвертированный индекс. Затем к этому множеству добавляются другие документы из того же кластера, даже если они слабо связаны с запросом.

Плоская кластеризация эффективна и проста, но имеет много недостатков. Алгоритмы плоской кластеризации создают простое неструктурированное множество кластеров, используя количество кластеров как входной параметр. Кроме того, эти алгоритмы являются недетерминированными. Иерархическая кластеризация создает иерархию, т.е. структурированное множество, которое является более информативным, чем неструктурированное множество кластеров, создаваемое в ходе плоской кластеризации. Для иерархической кластеризации не требуется заранее задавать количество кластеров, и большинство иерархических алгоритмов, используемых для информационного поиска, являются детерминированными. Однако за эти преимущества алгоритмов иерархической кластеризации приходится расплачиваться более низкой производительностью. Сложность наиболее распространенных алгоритмов иерархической кластеризации является как минимум квадратичной по отношению к количеству документов, в то время как алгоритм K-средних и EM-алгоритм имеют линейную сложность.

Алгоритмы иерархической кластеризации являются либо нисходящими, либо восходящими. Восходящие алгоритмы на начальном этапе сначала рассматривают каждый документ как отдельный кластер, а затем последовательно объединяют (или агломерируют) пары кластеров, пока они не сольются в один кластер, содержащий все документы. По этой причине восходящая иерархическая кластеризация называется агломеративной иерархической кластеризацией (HAC). Нисходящая иерархическая кластеризация основывается на разделении кластера. В ходе нисходящей кластеризации кластеры рекурсивно разделяются до тех пор, пока не будут расщеплены на отдельные документы. Восходящая иерархическая кластеризация используется для информационного поиска чаще, чем нисходящая. Именно она является предметом рассмотрения в данной главе.

Агломеративная иерархическая кластеризация обычно изображается с помощью дендрограмм, показанных на рис. 5.6. Каждое объединение представлено горизонтальной линией. Ордината этой горизонтальной линии

представляет собой меру сходства между двумя объединяемыми кластерами (отдельные документы рассматриваются как одноэлементные кластеры). Эта мера сходства называется комбинационной мерой сходства объединенного кластера. Например, комбинационная мера сходства кластера, состоящего из документов «Lloyd's CEO questioned» и «Lloyd's chief/ U.S. grilling» на рис. 5.6, примерно равна 0,56. Комбинационная мера сходства одноэлементного кластера трактуется как сходство документа с самим собой, или самоподобие (при использовании косинусной меры сходства оно равно единице).

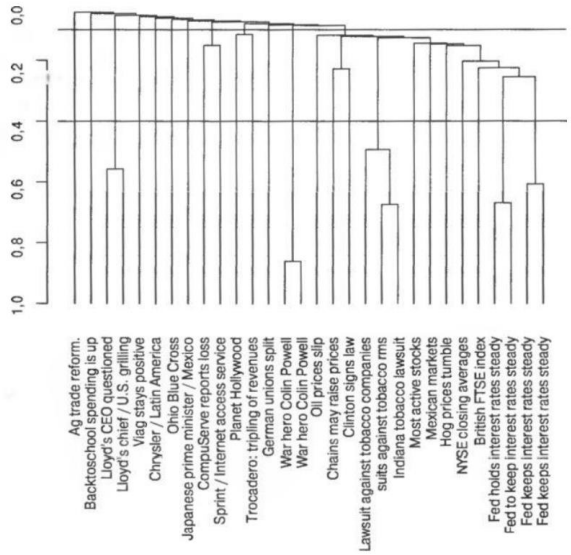


Рисунок 5.6 – Дендограмма кластеризации с одиночной связью 30 документов

Переходя по дендрограмме снизу вверх, можно проследить процесс кластеризации. Например, как показано на рис. 23, два документа с заголовками «War hero Colin Powell» были объединены первыми, а последнее объединение произошло, когда к кластеру, состоящему из 29 документов, был добавлен документ «Ag trade reform».

Основное предположение агломеративной иерархической кластеризации заключается в том, что операция объединения является монотонной. Это значит, что если s_1, s_2,..., s_{k-1} — комбинационные меры сходства последовательных объединений в агломеративной иерархической кластеризации, то выполняются неравенства $s_1 \geq s_2 \geq \ldots \geq s_{k-1}$. Немонотонная иерархическая кластеризация содержит хотя бы одну инверсию противоречащую основному предположению, что на каждом этапе выбирается наилучшее объединение.

При иерархической кластеризации не нужно задавать количество кластеров заранее. Однако в некоторых приложениях необходимо точно такое же разбиение на непересекающиеся кластеры, как и при плоской кластеризации. В этих ситуациях иерархию в определенной точке следует отсечь. Для определения точки отсечения существует много критериев.

— Отсечение на заранее указанном уровне сходства. Например, чтобы минимальное комбинационное сходство кластеров было равным 0,4, следует провести сечение дендрограммы на этом уровне. На рис. 23 показано, что сечение дендрограммы на уровне y = 0,4 порождает 24 кластера (группируются

только документы с высоким сходством), а сечение дендрограммы на уровне y = 0,1 порождает 12 кластеров (один крупный кластер финансовых новостей и одиннадцать более мелких кластеров).

— Сечение дендрограммы в точке максимальной разницы между двумя последовательными комбинационными мерами сходства. Такие перепады являются признаком «естественной» кластеризации. Добавление еще одного кластера в этом случае значительно ухудшает качество кластеризации, поэтому желательно делать отсечение до того, как это произойдет. Данная стратегия напоминает поиск точки перегиба на графике К-средних.

— Как и в плоской кластеризации, в иерархической кластеризации можно заранее задавать количество кластеров К и выбирать точку отсечения так, чтобы в итоге получить К кластеров.

2.1.6 Разложение матриц и латентно-семантическое индексирование

Рассмотрим разложения матриц «термин-документ», имеющих размерность M×N, где за редким исключением M≠N. Более того, маловероятно, что матрица C симметрична. Сначала рассмотрим расширение симметричного диагонального разложения, которое называется сингулярным. Связь между сингулярным разложением и симметричным диагональным разложением устанавливается теоремой «Пусть r — ранг матрицы C, имеющей размерность M×N. Тогда существует сингулярное разложение (SVD) матрицы $C=U\sum V^T$». Для данной матрицы C пусть U — матрица, имеющая размерность M×N, столбцы которой представляют собой ортогональные собственные векторы матрицы C^TC, а матрица V— матрица, имеющая размерность M×N, столбцы которой являются ортогональными векторами матрицы C^TC, где C^T — транспонированная матрица C.

Что же представляет собой матрица C^TC? Это квадратная матрица, строки и столбцы которой соответствуют M терминам. Элемент (i,j) этой матрицы является мерой перекрытия между i- и j-м терминами, основанной на их совместном появлении в документах. Точный математический смысл этого элемента зависит от способа определения весов терминов , с помощью которого построена матрица C. Рассмотрим вариант, в котором матрица C представляет собой матрицу инцидентности (см. рис. 5.7). В этом случае элемент (i,j) матрицы C^TC — это количество документов, в которых одновременно встречаются i- и j-й термины.

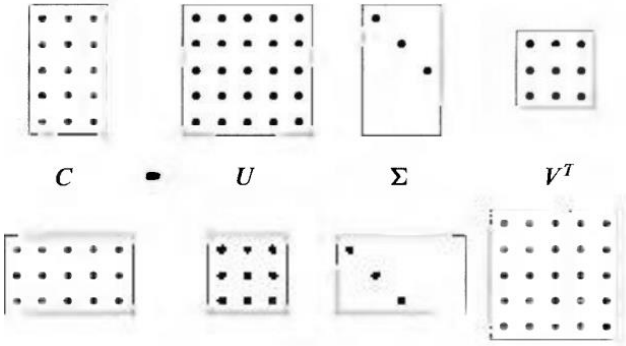


Рисунок 5.7 – Сингулярное разложение матрицы

Записывая сингулярное разложение, удобно выразить матрицу ∑ в виде матрицы r x r с сингулярными значениями на диагонали, поскольку все ее остальные элементы, в том числе диагональные при i > r, равны нулю Соответственно, можно проигнорировать M-r столбцов матрицы U, расположенных справа и соответствующих нулевым строками матрицы ∑. Аналогично можно проигнорировать правые N-r столбцов матрицы V, поскольку в матрице V^T они соответствуют строкам, которые умножаются на N-r нулевых столбцов матрицы ∑. Эту форму записи сингулярного разложения иногда называют **сокращенным сингулярным разложением** или **усеченным сингулярным разложением**.

Обсудим теперь приближение матрицы «термин-документ» C матрицей более малого ранга, построенной с помощью метода сингулярного разложения. Малоранговая аппроксимация матрицы C создает новое представление каждого документа в коллекции. Запросы также переводятся в малоранговое представление, что позволяет вычислять сходство между запросами и документами. Этот процесс называется **латентно-семантическим индексированием**.

Синонимия означает ситуацию, в которой два разных слова (например, car и automobile) имеют одно и то же значение. Векторное представление не позволяет учесть связь между синонимичными терминами, поэтому каждый из них порождает отдельную координатную ось векторного пространства. Вследствие этого вычисленная мера сходства (q,d между запросом q и документом d, недооцениват истинное сходство между ними, которое ищет пользователь. **Полисемия** возникает, когда термин, например charge, имеет несколько значений. В этом случае вычисленная мера сходства (q,d) переоценивает истинное сходство между запросом и документом, которое ищет пользователь.

Даже если коллекция имеет небольшой размер, матрица «термин-документ» C, как правило, содержит несколько десятков тысяч строк и столбцов, а ранг измеряется десятками тысяч. При латентно-семантическом индексировании (которое иногда называется латентно-семантическим анализом (LSA)) с помощью сингулярного разложения создается приближение $C_к$ матрицы «термин-документ», причем число k выбирается намного меньшим, чем ранг матрицы C размерности M×N независимо от числа k.

2.1.7 Основы поиска в вебе

Веб является беспрецедентным явлением по многим параметрам: по масштабу, по практически полному отсутствию координации при его создании, а также по разнообразию его участников. Каждая из этих особенностей отличает поиск в вебе от поиска «обычных» документов (и делает его намного сложнее). Изобретение гипертекста, предсказанного Ванневаром Бушем в 1940-х годах и впервые реализованного на практике в середин е 1970-х годов , предвосхитило процесс формирования Всемирной паутины в 1990-х годах. Веб рос гигантскими темпами и достиг уровня, когда его пользователями стала

значительная часть всего человечества. При этом сеть основана на простой открытой архитектуре «клиент-сервер»:

1) сервер взаимодействует с клиентом посредством простого протокола (http — hypertext transfer protocol, протокол передачи гипертекста), с помощью которого в асинхронном режиме передается разнообразная информация (текст, изображения , а со временем — аудио - и видеофайлы), закодированная с помощью простого языка разметки HTML (hypertext marku p language);

2) клиент, как правило, браузер — может игнорировать то, чего не понимает.

Огромная часть информации, размещенной в вебе, совершенно бесполезна до тех пор, пока она не обнаружена и воспринята пользователями. Первые попытки сделать информацию, размещенную в вебе, «обнаруживаемой» разбиваются на две категории: 1) полнотекстовые поисковые системы; 2) каталоги, систематизирующие веб-страницы по категориям. Первые предлагали интерфейс поиска, по ключевым словам, и были основаны на инвертированных индексах и механизмах ранжирования, аналогичных описанным ранее в этой книге. Вторые предоставляли пользователю возможность перемещаться по иерархическому дереву тематических меток.

Статическую часть веба, состоящую из статических HTML-страниц и гиперссылок между ними, можно представить в виде направленного графа, в котором каждый узел является веб-страницей, а каждое направленное ребро — гиперссылкой. На рис. 5.8 показаны два узла, A и B, принадлежащие графу.

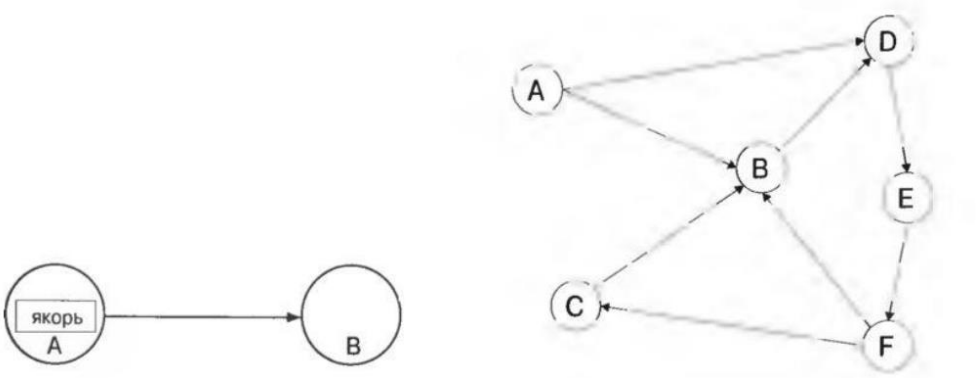


Рисунок 5.8 – Веб-граф

Каждый узел соответствует некоей веб-странице с гиперссылкой из A на B. Совокупность таких узлов и направленных ребер называется веб-графом. На рис. 5.8 показано также, что источник гиперссылки на странице A окружен текстом. Этот текст, как правило, инкапсулируется в атрибуте href тега, кодирующего гиперссылку в HMTL-коде страницы A, и называется текстом ссылки. Можно предположить, что этот направленный граф не является сильно связанным, поскольку существуют такие пары страниц, что с одной из них невозможно перейти на другую, следуя по гиперссылкам. Гиперссылки, указывающие на страницу, называются входящими, а указывающие вовне — исходящими. Количество входящих ссылок на страницу в среднем колеблется

от 8 до 15. Аналогично определяется полустепень исхода как количество исходящих ссылок. Эти понятия проиллюстрированы на рис. 5.8.

Всем знакомо явление спама. Причиной спама является неоднородность мотивов создания веб-страниц. В частности, многие создатели веб-страниц руководствуются коммерческими интересами и, следовательно, стремятся достичь выгоды за счет манипулирования результатами поиска. На это можно возразить, что спам ничем не отличается от ситуации, когда компании печатают номера своих телефонов крупным шрифтом на «желтых страницах» рекламных справочников. Однако это стоит компании больших затрат и, следовательно, является более честным способом рекламирования. Возможно, более удачная аналогия — использование в качестве названия компаний длинных строк, начинающихся с нескольких букв "А", чтобы попасть на первые места списка в соответствующей категории рекламного справочника. Фактически структура рекламных справочников, в которых компании, желающие выделиться за счет более крупного или более темного шрифта, должны заплатить больше, была воспроизведена и в веб-поиске. Многие поисковые системы допускают возможность дополнительной оплаты за включение веб-страницы в свой индекс. Эта модель называется платным включением. Разные поисковые системы придерживаются разных правил, касающихся оплаченного включения и влияния этой оплаты на ранжирование результатов.

2.1.8 Обход и индексирование веба

Обход веба (web crawling) — это сбор страниц в вебе для их дальнейшего индексирования и поддержки функционирования поисковой системы. Цель обхода — быстро и эффективно собрать как можно больше полезных веб-страниц вместе со ссылками, которые их объединяют. Ранее мы говорили о сложности веба, причина которой в том, что в создании сети принимают участие миллионы независимых индивидуумов. В настоящей главе мы исследуем проблемы, возникающие при обходе веба. В центре внимания здесь находится компонент — поисковый робот (web crawler или spider).

Требования к поисковым роботам разделяются на две категории: обязательные и желательные.

Устойчивость. В вебе существуют серверы, создающие ловушки для поисковых роботов. Он и генерируют веб-страницы, вынуждающие поисковых роботов зацикливаться на бесконечном количестве веб-страниц в определенном домене. Поисковые роботы должны быть устойчивыми к таким ловушкам. Не все ловушки созданы злонамеренно; некоторые из них возникают как побочные эффекты небрежной разработки веб-сайта.

Вежливость. Веб-серверы могут иметь явные и неявные правила, регулирующие частоту обращений поисковых роботов. Следует соблюдать эти правила.

Желательные свойства поискового робота:

— распределенность – желательно, чтобы поисковый робот имел возможность функционировать в распределенном режиме на многих машинах;

— масштабируемость – желательно, чтобы архитектура поискового робота допускала повышение производительности обхода путем добавления новых машин и расширения полосы пропускания;

— производительность и эффективность – желательно, чтобы поисковый робот эффективно использовал разнообразные ресурсы поисковой системы, включая процессор, память и полосу пропускания компьютерной сети;

— качество – учитывая, что значительная часть веб-страниц плохо удовлетворяет информационные потребности пользователей, желательно, чтобы поисковый робот при обходе отдавал преимущество, в первую очередь, полезным страницам;

— свежесть – во многих приложениях желательно, чтобы поисковый робот работал непрерывно, получая свежие копии ранее загруженных страниц.

— расширяемость – поисковые роботы следует разрабатывать так, чтобы их можно было расширять по разным направлениям— учитывать новые форматы данных, новые протоколы передачи данных и т.д.

Основное назначение поискового робота заключается в следующем:

1. Поисковый робот начинает работу с одного или нескольких URL, образующих начальное множество.
2. Он извлекает URL из этого множества, а затем загружает по нему веб-страницу.
3. После этого выбранная страница обрабатывается и из нее извлекаются текст и ссылки (каждая из них указывает на другой URL).
4. Извлеченный текст передается индексатору. Извлеченные ссылки (URL) передаются очереди на скачивание URL, состоящей из URL, соответствующих страницам, которые должны быть загружены поисковым роботом. В самом начале очередь на скачивание URL содержит начальное множество; как только страница выбрана, ее адрес удаляется из очереди на скачивание URL.
5. В целом этот процесс можно рассматривать как обход веб-графа.
6. При непрерывном обходе URL выбранной страницы добавляется обратно в очередь URL, чтобы загрузить ее еще раз в будущем.

Этот внешне простой рекурсивный обход веб-графа усложняется многочисленными требованиями, предъявляемыми к поисковым роботам: поисковый робот должен быть распределенным, масштабируемым, корректным, устойчивым и расширяемым, причем выбранные страницы должны иметь высокое качество.

2.1.9 Анализ ссылок

В развитии веб-поиска важную роль играет анализ гиперссылок и структуры веб-графа.

Вспомните понятие веб-графа. Изложение методов анализа ссылок основывается на двух интуитивных предположениях.

1. Текст ссылки, указывающей на страницу B, является хорошим описанием страницы B.

2. Гиперссылка со страницы A на страницу B представляет собой признание авторитетности страницы B со стороны создателя страницы A. Это не всегда так; например, многие ссылки со страницы на страницу внутри одного веб-сайта обязаны своим появлением общему шаблону сайта. Например, большинство корпоративных веб-сайтов имеют на каждой странице ссылки на уведомление об авторском праве. Совершенно ясно, что это не является свидетельством одобрения. Соответственно, алгоритмы анализа ссылок учитывают такие ссылки с меньшим весом.

Нередко между терминами на веб-странице и описанием страницы пользователем существует несогласованность. Значит, при поиске в вебе нет необходимости запрашивать термины, находящиеся на странице. Добавим что многие веб-страницы заполнены графическими изображениями. При анализе таких HTML-страниц невозможно извлечь полезный для индексирования текст. Для решения проблемы, как вариант, можно можно использовать текст ссылок, воспользовавшись знаниями сообщества авторов веб-страниц.

Реальные веса терминов определяются с помощью взвешивания на основе метода машинного обучения. По-видимому, современные поисковые машины приписывают существенный вес терминам ссылок.

Рассмотрим теперь методы вычисления весов и ранжирования, вытекающие исключительно из структуры ссылок. Первый метод анализа ссылок присваивает каждому узлу веб-графа вес в диапазоне от нуля до единицы, известный как PageRank. Вес узла Page Rank зависит от структуры ссылок в веб-графе. По заданному запросу поисковая машина вычисляет итоговый вес каждой веб-страницы, учитывающий сотни разных признаков, например косинусную меру близости и близость терминов в сочетании с весом PageRank. Этот составной вес, вычисленный с помощью методов, используется для создания списка ранжированных результатов поиска по запросу.

2.1.10Вопросы к лекции №5

1 В чем суть наивного байесовского подхода при классификации текстов.

2 Опишите модель информационного поиска Бернулли.

3 Каковы представление документов и меры близости в векторном пространстве.

4 Опишите механизм автоматической категоризации текстов (метрический метод Роккио).

5 Опишите метод k ближайших соседей и его применение в информационном поиске.

6 Опишите линейные и нелинейные классификаторы и их применение в информационном поиске.

7 Опишите метод опорных векторов: случай линейно разделимых классов и его применение в информационном поиске.

8 Назовите методы машинного обучения для поиска по запросу.

9 В чем суть кластеризации в информационном поиске.

10 Опишите метод k средних и его применение в информационном поиске.

11 В чем суть кластеризации, основанной на моделях.

12 Назовите и опишите основные типы иерархической кластеризации.

13 Опишите латентно-семантическое индексирование.

14 Для чего служат мало ранговые аппроксимации в информационном поиске и обработке естественного языка?

2.2 Лекция №6 Методы обработки естественных языков

План лекции№6

1. Анализ тональности
2. Нечеткий поиск дубликатов
3. Метод N-грамм
4. Распознавание имён людей, географических названий и других сущностей
5. Мешок слов

Обработка естественного языка (Natural Language Processing, или NLP) — это направление в исследовании искусственного интеллекта, задача которого — обучить машину понимать и обрабатывать язык человека. Разберем основные методы обработки естественных языков и особенности их использования.

2.2.1 Анализ тональности

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".

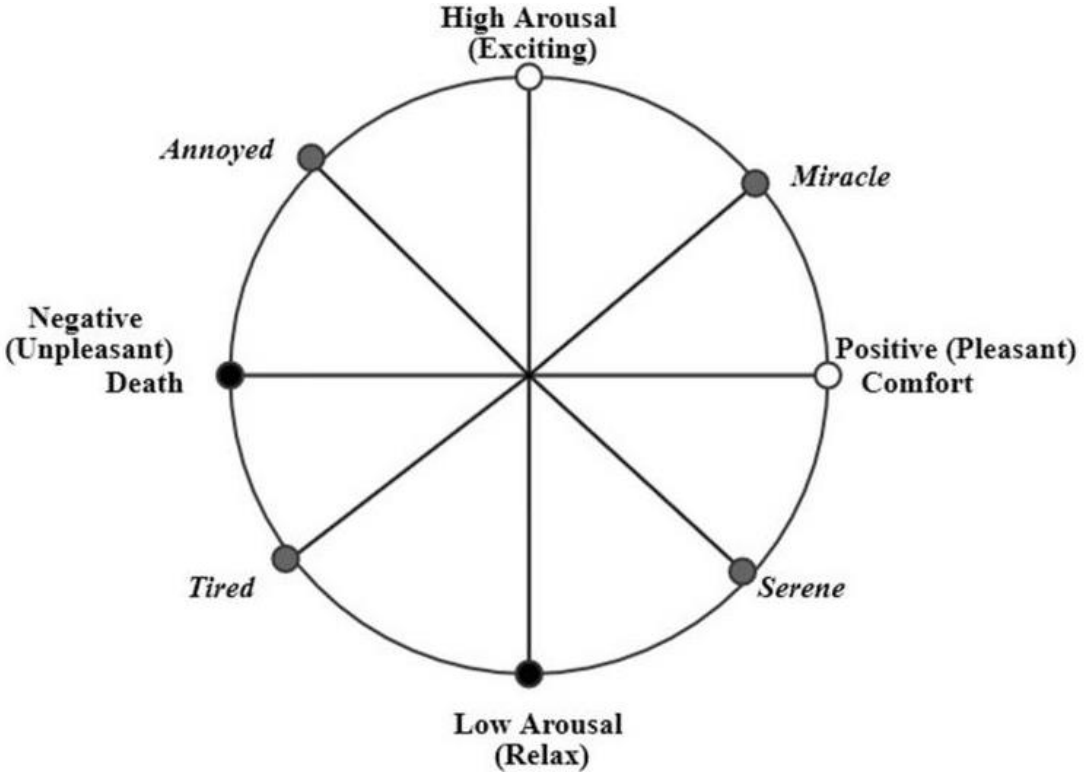


Рисунок 6. 1 – Размерная модель эмоций

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_seq_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_dropout=0.2)))
model.add(Dense(num_classes, activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

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,validation_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.

2.2.2 Нечеткий поиск дубликатов

Инструмент для выявления плагиата w-shingling - алгоритм шинглов (от англ. shingles — чешуйки) — алгоритм, разработанный для поиска копий и дубликатов рассматриваемого текста в веб-документе.

В естественном языке обработка в алгоритм шинглова представляет собой набор уникальной черепицы (поэтому н-грамм) , каждый из которых состоит из смежных подпоследовательностей из маркеров в пределах документа , которые затем могут быть использованы для установления подобия между документами . Символ w обозначает количество токенов в каждом выбранном или решенном шингле.

Таким образом, документ *"a rose is a rose is a rose"* можно максимально токенизировать следующим образом:*(a,rose,is,a,rose,is,a,rose)*

Множество всех смежных последовательностей 4 маркеров (таким образом 4 = п , таким образом 4- г) является

{ (a,rose,is,a), (rose,is,a,rose), (is,a,rose,is), (a,rose,is,a), (rose,is,a,rose) }

Которая затем может быть уменьшена или максимально покрыта черепицей в данном конкретном случае до *{ (a,rose,is,a), (rose,is,a,rose), (is,a,rose,is) }.*

Уди Манбер . в 1994 г. первым в мире выразил идею поиска дубликатов, а в 1997 г. Андрей Бродер оптимизировал и довел ее до логического завершения, дав имя данной системе — «алгоритм шинглов».

Реализацию алгоритма рассморим на практическом занятии, описанную Зеленковым Ю. Г. и Сегаловичем И.В. в публикации «Сравнительный анализ методов определения нечетких дубликатов для Web-документов». Предлагается версия алгоритма шинглов, использующая случайную выборку 84х случайных шинглов.

Почему именно 84? Использование 84х случайно выбранных значений контрольных сумм позволит легко модифицировать алгоритм до алгоритма супершинглов и мегашинглов, которые гораздо менее ресурсоемки.

2.2.3 Метод N-грамм

N-грамма — последовательность из n элементов (последовательность звуков, слогов, слов, букв). N-грамма может быт представлена как ряд слов. При этом устойчивые словосочетания называют коллокацией.

Биграмма — последовательность из двух последовательных элементов, из трех элементов — **триграмма**. Из четырех и выше элементов — N-грамма. N заменяется на количество последовательных элементов.

Применяются N-граммы в области теоретической математики, биологии, картографии, в музыке. Использование N-грамм связано с процессами – извлечения данных для кластеризации, например, серии спутниковых снимков Земли из космоса, поиска генетических последовательностей, для индексирования данных в поисковых системах, в т.ч. данных, связанных со звуком.

N-граммы используется в NLP для предугадывания на основе вероятностных моделей. Модель рассчитывает вероятность последнего слова N-граммы, если известны все предыдущие. Методика основана на том, что

N-граммы служат для проверки текста на оригинальность. Тест делится текст на несколько обозримых частей, для сравнения в виде N-граммам. В результате анализируется степень сходства документов.

Кроме того, N-граммы часто успешно используются для категоризации текста и языка, для создания функций, которые позволяют получать знания из текстовых данных, замены слова ошибочных слов т.д.

N-граммная модель основана на определении вероятности употребления заданной фразы. Эту вероятность формально можно задать как вероятность возникновения последовательности слов в неком корпусе. Например, вероятность фразы *«искренность есть общение без фальши»* можно вычислить как произведение вероятностей каждого из слов этой фразы:

*P = P(искренность) * P(есть|искренность) * P(общение|искренность есть) * P(без|искренность есть общение) * P(фальши|искренность есть общение без)*

Чтобы определить *P(искренность)*, нужно посчитать, сколько раз это слово встретилось в тексте, и поделить это значение на общее число слов. Для того чтобы рассчитать вероятность *P(фальши|искренность есть общение без)* нужно упростить эту задачу. Пусть вероятность слова в тексте зависит только от предыдущего слова. Тогда наша формула для расчета фразы примет следующий вид:

*P = P(искренность) * P(есть|искренность) * P(общение|есть) * P(без|общение) * P(фальши|без)*

. Для рассчета условной вероятности *P(есть|искренность)* посчитаем количество пар 'искренность есть', и делим на количество в тексте слова 'искренность'.

В результате, если мы посчитаем все пары слов в некотором тексте, мы сможем вычислить вероятность произвольной фразы. Этот набор рассчитанных вероятностей и будет биграммной моделью.

N-граммные модели востребованы исследовательскими центрами Google. На основании этих моделей разработаны многие проекты: статистический перевод с одного языка на другой, распознавание речи, исправление орфографических ошибок, извлечение информации и т.д. При этом были использованы корпусы, содержащие несколько триллионов слов. Создан учебный корпус Google teracorpus. Он содержит 10^{12} слов.

Рассмотрим алгоритм для извлечения N-грамм из текста. Этот алгоритм должен работать с неограниченным размером текста, быстро и эффективно использовать имеющиеся ресурсы. Известные методы извлечения N-грамм из текста представлены на рисунке 6.2. Они основаны на разных принципах.

Различают синтаксические и линейные N-граммы. Синтаксические N-граммы — это N-граммы, определяемые путями в деревьях синтаксических зависимостей или деревьях составляющих, а не линейной структурой текста. Например, предложение: «Политические новости оказывают негативное влияние на» может быть преобразовано в синтаксические N-граммы, следуя древовидной структуре его отношений зависимостей: новости-политические, влияние-негативное, влияние-на-рынки-финансовые и другие.

Синтаксические N-граммы представляют синтаксическую структуру в отличие от линейных N-грамм. Синтаксические N-граммы могут использоваться в тех же приложениях, что и линейные N-граммы, в том числе в качестве признаков в векторной модели.

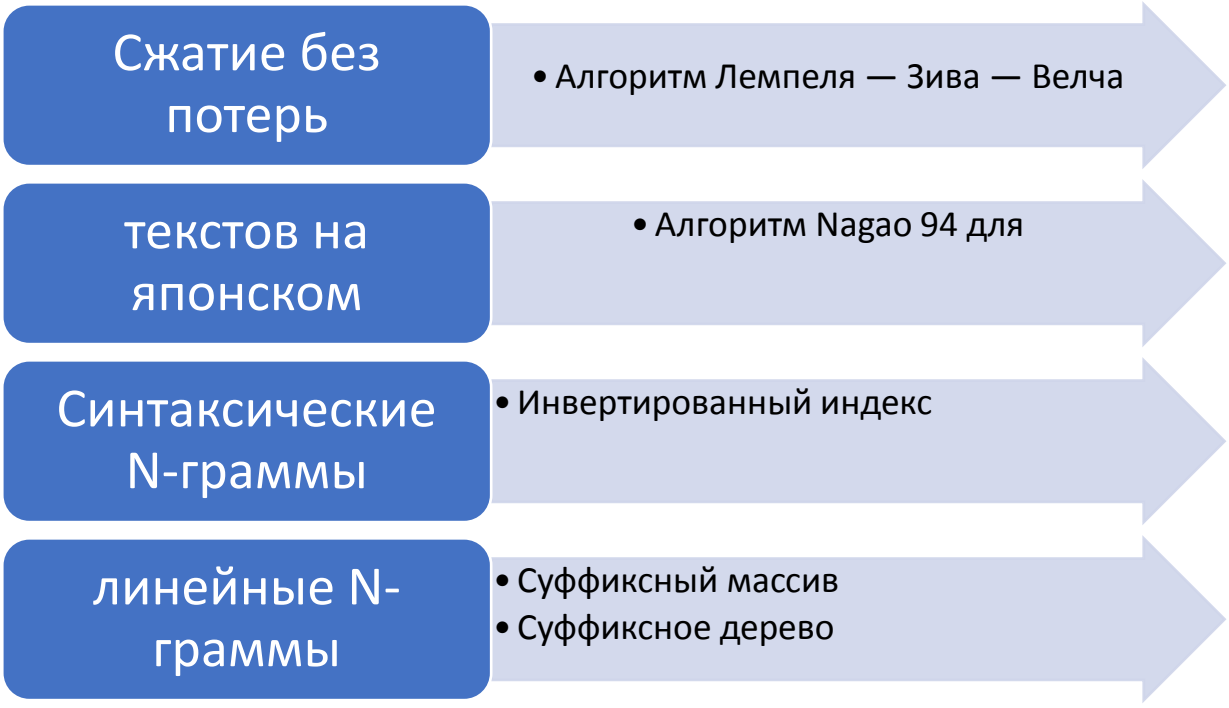


Рисунок 6.2- Состав методов извлечения N-грамм

Применение синтаксических N-грамм дает лучшие результаты при решении определенных задач, чем использование стандартных N-грамм, например, для определения авторства.

2.2.4 Распознавание имён людей, географических названий и других сущностей

Имена людей, названия организаций, книг, городов, и другие имена собственные называют «именованные сущности» (named entities), а саму задачу — «распознавание именованных сущностей». По-английски «Named entity recognition» или коротко NER; это сокращение регулярно используется и в русскоязычных текстах.

За одной задачей NER, на самом деле, стоит две:

1) обнаружить, что какая-то последовательность слов — это именованная сущность;

2) понять, к какому классу (имя человека, название организации, город и т.п.) эта именованная сущность относится. На каждом из этапов возникают свои сложности.

Начнем со второго этапа: какие классы именованных сущностей обычно хотят найти. Никакого стандартного набора классов нет. Практически всегда стараются извлекать имена людей и названия мест и организаций, а дальше все зависит от конкретных задач, которые нужно решать, или от возможностей предобученной системы, которую планируется использовать. Также к задаче NER относят извлечение дат, денежных сумм (число + валюта), которые интуитивно не выглядят как именованные сущности.

Многие именованные сущности могут в разных контекстах относиться к разным классам: слово «Пушкин» может быть человеком, городом, названием клуба и т.д. Понять, к какому именно классу слово относится в конкретном контексте, — сложная задача, которой сейчас много занимаются.

На первый взгляд кажется, что с именованными сущностями не должно возникнуть особых проблем, большинство из них -— имена собственные, они пишутся с большой буквы, так что найти их в тексте не сложно. Во-первых, во многих языках вообще нет больших букв (например, в китайском или арабском). Но и в европейских языках приходится сталкиваться с трудностями. Основные сложности:

Именованные сущности редко состоят из одного слова. Например, из предложения «Звонил доктор ВинеаминГумгард» нужно извлечь, как минимум, имя и фамилию — «ВинеаминГумгард», а для многих задач полезно уметь находить полное «наименование» человека, о котором говорится: «директор ВинеаминГумгард». Аналогично и с другими именованными сущностями: «Министерство просвещения Российской Федерации», «19 октября», «100 000 динаров» и т.п.

Не всегда очевидны границы, например, словосочетание «Иван Васильевич и партнеры» может быть названием некоторой организации, а может подразумевать некоторого Ивана Васильевича и отдельно его партнеров.

С большой буквы пишутся не только имена собственные. Это верно и для русского, и еще больше для английского, но особенно хорошо видно в немецком, где с большой буквы пишутся вообще все существительные.

Механизм работы. Большинство именованных сущностей состоит из нескольких слов, при решении этой задачи обычно рассматривают отдельные слова и решают, является ли это слово частью именованной сущности или нет. При этом различают начало, середину и конец именованной сущности.

При разметке именованных сущностей принято использовать префикс «B» (beginning) для обозначения первого слова, «E» — для последнего слова и «I» (intermediate) — для всех слов между. Иногда также используется префикс «S» (single) для обозначения именованной сущности, состоящей из одного слова.

Таким образом, задача сводится к пословной классификации. Сейчас для построения классифкатора обычно тренируют нейросетевую модель на большом количестве текстов с разметкой именованных сущностей. В основе моделей обычно стоят модули, которые позволяют учитывать контекст с двух сторон от анализируемого слова (например, bi-LSTM или bi-RNN).

Хорошие результаты дают и классические классификаторы, работающие с предзаданным множеством признаков. Большие буквы или нестандартное использование больших и маленьких букв (iPhone), а также специфических символов (H&M) внутри слова — все это полезныепризнаки для выявления именованных сущностей.

Также до сих пор используются системы, основанные на правилах: в них прописываются разные шаблонные схемы именованных сущностей. Например, шаблон «Министерство + образования\туризма\здравоохранения + Название страны» поможет найти «Министерство образования Италии» и «Министерство туризма Мексики». Шаблон «однозначное\двузначное число + название месяца + четырехзначное число» — это дата. Основная проблема такого подхода: на подготовку правил требуется очень много времени, а малейшее отступление от паттерна (например, банальная опечатка) все сломает. Также набор правил пишется под определенный язык, поэтому набор правил, помогающий извлекать именованные сущности из текстов на русском языке никак не поможет при обработке другого языка.

Активно используемых именованных сущностей конечное количество, многие из них уже собраны в списки (так называемые газетиры, gazetteers): географические названия, словари имен, реестры организаций и т.п. Всю эту информацию тоже стараются использовать для решения задачи NER..

2.2.5 Модель мешка слов

Модель набора слов — это упрощенное представление, используемое при обработке естественного языка и поиске информации (IR). В этой модели текст (например, предложение или документ) представлен как мешок (мультимножество) своих слов, без учета грамматики и даже порядка слов, но с сохранением множественности. модель мешка слов также использовалась для компьютерного зрения.

Модель набора слов обычно используется в методах классификации документов, где (частота) появления каждого слова используется в качестве признака для обучения классификатора.

Раннее упоминание «мешка слов» в лингвистическом контексте можно найти в статье Зеллига Харриса 1954 года о структуре распределения.

Модель мешка слов — это способ представления текстовых данных при моделировании текста с помощью алгоритмов машинного обучения.

Модель «мешок слов» проста для понимания и реализации и имеет большой успех в таких проблемах, как моделирование языка и классификация документов.

Модель мешка слов, или сокращенно BoW, — это способ извлечения особенностей из текста для использования в моделировании, например, в алгоритмах машинного обучения.

Подход очень прост и гибок, и его можно использовать множеством способов для извлечения функций из документов.

Мешок слов — это представление текста, который описывает вхождение слов в документ. Это включает в себя две вещи:

1. Словарь известных слов.
2. Мера наличия известных слов.

Это называется «мешок слов, потому что любая информация о порядке или структуре слов в документе отбрасывается. Модель проверяет только то, встречаются ли в документе известные слова. Гипотеза в том, что что документы похожи, если они имеют похожее содержание.

Мешок слов может быть простым или сложным. Сложность заключается как в определении словарного запаса известных слов (или токенов), так и в том, как оценивать наличие известных слов.

Все упорядочения слов номинально отбрасываются, и у нас есть последовательный способ извлечения функций из любого документа в нашем корпусе, готового для использования в моделировании.

Новые документы, которые пересекаются со словарем известных слов, но могут содержать слова вне словаря, все еще могут быть закодированы, где только вхождение известных слов оценивается, а неизвестные слова игнорируются.

Вы можете видеть, как это может естественным образом масштабироваться до больших словарей и больших документов.

Пример реализации

Следующее моделирует текстовый документ с использованием набора слов . Об этом говорит сайт https://intellect.icu . Вот два простых текстовых документа:

(1) Джон любит смотреть фильмы. Мэри тоже любит фильмы.

(2) Мэри также любит смотреть футбольные матчи.

На основе этих двух текстовых документов для каждого документа составляется список следующим образом:

"Джон" , "любит" , "смотреть" , "фильмы" , "Мэри" , "любит" , "фильмы" , "тоже"

«Мэри» , «также» , «любит» , «на» , «смотреть» , «футбол» , матчи»

Представление каждого пакета слов как объект JSON и присвоение соответствующей переменной JavaScript :

BoW1 = { "Джон" : 1 , "нравится" : 2 , " кому " : 1 , "смотреть" : 1 , "фильмы" : 2 , "Мэри" : 1 , "тоже" : 1 };

BoW2 = { "Мэри" : 1 , "также" : 1 , "нравится" : 1 , " кому " : 1 , "смотреть" :: 1 , «игры» : 1 };

Каждый ключ - это слово, а каждое значение - это количество вхождений этого слова в данный текстовый документ.

Порядок элементов свободный, поэтому, например {"too":1,"Mary":1,"movies":2,"John":1,"watch":1,"likes":2,"to":1}, также эквивалентен BoW1 . Это то, чего мы ожидаем от строгого представления объекта JSON .

Примечание: если другой документ подобен объединению этих двух,

(3) Джон любит смотреть фильмы. Мэри тоже любит фильмы. Еще Мэри любит смотреть футбольные матчи.

его представление в JavaScript будет:

BoW3 = { "Джон" : 1 , "нравится" : 3 , " кому " : 2 , "смотреть" : 2 , "фильмы" : 2 , "Мэри" : 2 , "тоже" : 1 , "также" : 1 , "футбол" : 1 , "игры" : 1 };

Итак, как мы видим в алгебре мешков, «объединение» двух документов в представлении мешков слов формально является несвязным объединением, суммирующим кратности каждого элемента.

Управляющий словарь

По мере увеличения словарного запаса увеличивается и векторное представление документов. В рассмотренном примере длина вектора документа равна количеству известных слов.

2.2.6 Суп из тегов

В веб-разработке « суп из тегов » - это уничижительное слово для синтаксически или структурно некорректного HTML, написанного для веб-страницы . Поскольку веб-браузеры исторически снисходительно относились к синтаксису HTML или структурным ошибкам, веб-разработчики не оказывали большого давления на соблюдение опубликованных стандартов, и поэтому существует необходимость во всех реализациях браузеров обеспечивать механизмы, позволяющие справиться с появлением «супа тегов», принятие и исправление неверного синтаксиса и структуры, где это возможно. Анализатор HTML (часть веб-браузера), способный интерпретировать разметку, подобную HTML, даже если он содержит недопустимый синтаксис или структуру, может называться парсером супа тегов .

В настоящее время все основные веб-браузеры имеют парсер тегов для интерпретации искаженного HTML, при этом большинство элементов обработки ошибок стандартизированы. «Суп тегов» включает в себя множество распространенных ошибок при создании , таких как искаженные теги HTML , неправильно вложенные элементы HTML и неэкранированные символьные объекты (особенно амперсанды (&) и знаки «меньше» (<)). Я использовал этот термин в своих инструкциях на протяжении многих лет, чтобы охарактеризовать беспорядок угловых скобок, действующих как теги в HTML на страницах, которые принимаются браузерами. Неправильная минимизация,

перекрывающиеся конструкции ... все, что выглядит как разметка SGML, но создатель не знал и не соблюдал правила SGML для словаря HTML. По сути, объемный набор текста и разметки. [...] Я никогда нигде не видел определение этого термина. - Г. Кен Холман, Re: [xml-dev] Что такое Tag Soup? , Список рассылки разработчиков XML, 11 октября 2002 г. Служба проверки разметки - это ресурс для авторов веб-страниц, позволяющий избежать создания супа тегов.

«Суп из тегов» - это термин, используемый для очернения различных методов веб-разработки. Некоторые из них (в порядке убывания степени тяжести) включают:

Неправильная разметка, когда теги неправильно вложены или неправильно закрыты. Например, следующее: This is a malformed fragment of HTML.

Недопустимая структура, в которой элементы неправильно вложены в соответствии с DTD для документа. Примеры этого включают вложение элемента «ul» непосредственно внутри другого элемента «ul» для любого из HTML 4.01 или XHTML DTD. Dan Connolly приводит использование заголовка элемента вне головной части.

Использование собственных или неопределенных элементов и атрибутов вместо тех, которые определены в рекомендациях W3C. Например, использование элемента Blink или элемента Marquee, которые были нестандартными элементами, изначально поддерживались только браузерами Netscape и Internet Explorer соответственно.

Спецификация XML четко определяет, что соответствующий пользовательский агент (например, веб-браузер) не должен принимать документ и не продолжать его анализ, если обнаружена какая-либо синтаксическая ошибка. Таким образом, браузер, интерпретирующий веб-страницу как XHTML, откажется отображать страницу, если обнаружит ошибку формирования. Это может помочь гарантировать, что, когда авторы тестируют код XHTML в соответствующем браузере, они сразу же будут проинформированы о проблемах с искажениями: возможно, это самая серьезная проблема, с которой сталкиваются веб-браузеры.

Когда код искажен, намерения автора неоднозначны. Без директив XML HTML-браузеры должны использовать сложные алгоритмы для вывода предполагаемого автором значения в широком диапазоне случаев, когда встречается недопустимый синтаксис. XML и XHTML представляют концепцию пространств имен. С помощью пространств имен авторы или сообщества авторов могут определять новые элементы и атрибуты с новой семантикой и смешивать их в своих документах XHTML. Пространства имен гарантируют, что имена элементов из различных пространств имен не будут объединены. Например, элемент «таблица» может быть определен в новом пространстве имен с новой семантикой, отличной от элемента «таблица» HTML, и браузер сможет различать эти два элемента. Предоставляя пространства имен, XHTML в сочетании с CSS позволяет сообществам разработчиков легко расширять семантический словарь документов. Это

позволяет использовать проприетарные элементы при условии, что эти элементы могут быть представлены целевой аудитории через полные определения таблиц стилей (включая звуковые / речевые и тактильные стили). Документы XHTML могут обслуживаться в Интернете с использованием типа Интернет-носителя, application/xhtml+xml или text/html текущие версии Microsoft Internet Explorer (6, 7 и 8) не отображают документы XHTML, в качестве которых используются application/xhtml+xml . Бета-версии IE9 кажутся совместимыми. См. Также обсуждение этого вопроса в статье XHTML . HTML5 Основная статья: HTML5 HTML5 на данный момент стремится быть наиболее полным решением проблемы супа тегов, оставаясь при этом максимально совместимым с предыдущими и последующими версиями. В отличие от XHTML, который отходит от обратной совместимости и использует подход, согласно которому парсеры должны стать менее терпимыми к плохо сформированной разметке, HTML5 признает, что плохо сформированный код HTML уже существует в больших количествах и, вероятно, будет продолжать использоваться, и считает, что спецификацию следует расширить, чтобы обеспечить максимальную совместимость с таким кодом. Таким образом, спецификация HTML 5 изменила свое определение синтаксиса HTML, чтобы приспособить общий синтаксис, используемый сегодня, и явно описать, как именно "плохо сформированный код" должен обрабатывать синтаксический анализатор. Обработка плохо сформированного кода теперь имеет место в самой спецификации, что, как мы надеемся, снижает потребность в будущих анализаторах HTML для реализации дополнительных мер, выходящих за рамки спецификации, для работы с кодом, который он не распознает. Инструменты для исправления тегов супа HTML Tidy - это программный инструмент, доступный для многих платформ, который может исправлять недопустимый синтаксис и наиболее недопустимую структуру документа, преобразовывая HTML-подобный код в HTML или XHTML. Aggiorno - это надстройка Visual Studio, предназначенная для обеспечения соответствия веб-сайтов стандартам. TagSoup - это библиотека Java, которая анализирует HTML, очищает его и доставляет поток событий SAX, представляющих правильно сформированный XML (не обязательно действительный XHTML). Эти инструменты используются для обработки файлов JNLP в реализации протокола JNLP с открытым исходным кодом , доступного в IcedTea-Web , подпроекте IcedTea , проекте сборки и интеграции OpenJDK . Beautiful Soup - это синтаксический анализатор, похожий на Python DOM, для бессмысленного HTML / XML. tagsoup : библиотека для языка Haskell . QUANTUM SYSTEM 20-летняя богачка взорвала г. Краснодар, показав схему обогащения Такого еще не было. Смотри и запоминай все хитрости УЗНАТЬ БОЛЬШЕ→ Допустимые отклонения от XHTML В отличие от строгого XHTML, HTML и его предшественник SGML предназначены для написания людьми и уже обладают значительной степенью гибкости в синтаксисе для сокращения шаблонов. Эти различия не делают документ недействительным и, следовательно, не являются супом тегов. Следующее применимо как к HTML 4, так и к HTML5, и примеры относятся к первым дням HTML. Теги вроде <head>...</head> часто можно

полностью опустить. Замыкание тегов часто может быть опущено, потому что спецификация отклоняет некоторые элементы, вложенные в себя. Например, <li>... можно записать несколько элементов без закрытия. Несмотря на их обоснованность, эти упущения по-прежнему требуют специального синтаксического анализатора со знанием HTML (в отличие от более жесткого XML) для анализа. Кроме того, обычно инструменты «исправляют» и эти структуры. Например, tidy позволяет опускать необязательные теги, но по умолчанию этого не делает.

2.2.7 Вопросы к лекции №6

1. Расскажите о методике нечеткого поиска дубликатов.
2. В чем суть метод N-грамм и в каком случае целесообразно его применение?
3. Дайте понятие определения корпус текстов.
4. Расскажите о методике использования корпуса текстов
5. Каким образом можно произвести распознавание имён людей, географических названий и других сущностей
6. Расскажите о методике использования метода «Мешок слов».
7. Поясните смысл слов «Суп из тегов»

2.3 Лекция №7 Вопросно-ответные системы

План лекции № 7

1. Определение понятия вопросно-ответной системы и механизм их работы.
2. Назначение вопросно-ответных систем.
3. Классификация вопросно-ответных систем.
4. QA-система Start.
5. Тернарное выражение.

2.3.1 Определение понятия вопросно-ответной системы и механизм их работы

В настоящее время активно вопросно-ответные системы (Question-Answering Systems, QAS). Они основаны на извлечении информации непосредственно из документа, разговора, онлайн поиска и т.п.

Вопросно-ответная система — информационная система, способная принимать вопросы и отвечать на них на естественном языке.

Впервые вопросно-ответные системы применены в 1960-х годах как оболочки для экспертных систем. Современные системы предназначаются для поиска ответов на различные вопросы и используют технологий обработки естественных языков.

В отличие от задачи классического поиска по ключевым словам, в которой результатом является перечень документов, содержащих ответ на вопрос, в задаче вопросно-ответного поиска результат – это краткий и лаконичный ответ, сформированный системой в результате анализа разнообразных источников данных.

Примером такого источника может служить некоторая коллекция полнотекстовых документов (множество страниц глобальной сети Интернет), а ответ составляется из фрагмента наиболее релевантного документа коллекции. Традиционно в работах по вопросно-ответному поиску приводят классификацию методов или систем по используемому математическому аппарату:

— логические формы и логический вывод,
— графы зависимостей слов в предложениях,
— статистический подход и машинное обучение классификаторов,
— лексические онтологии для анализа отдельных слов текста и др.

QA системы позволяют пользователю не читать полный текст, а изучать подобранные автоматически короткие и лаконичные ответы. Они комбинируются с чат-ботами и извлекают информацию из набора картинок.

Вопросно-ответная система – это конгломерат поисковых, справочных и интеллектуальных систем. При этом она использует естественно-языковой интерфейс. На рисунке 7.1 изображена обобщенная схема работы вопросно-ответной системы

Из этой схемы видно, что **подаётся запрос** на естественном языке, он обрабатывается и генерируется ответ также на естественном языке. При этом применяется следующая схема: сначала система отбирает документы,

содержащие информацию, связанные с поставленным вопросом, затем фильтрует их, выделяя отдельные текстовые фрагменты, потенциально содержащие ответ, после чего из отобранных фрагментов синтезирует ответ на вопрос.

Источником информации вопросно-ответной системы служит локальное хранилище, или глобальная сеть, или и то и другое одновременно. Информация в Интернете неструктурированная и для её корректного извлечения необходимо создание оберток. «Обертока» (wrapper) - подпрограмма для унифицированного доступа к различным информационным ресурсам.

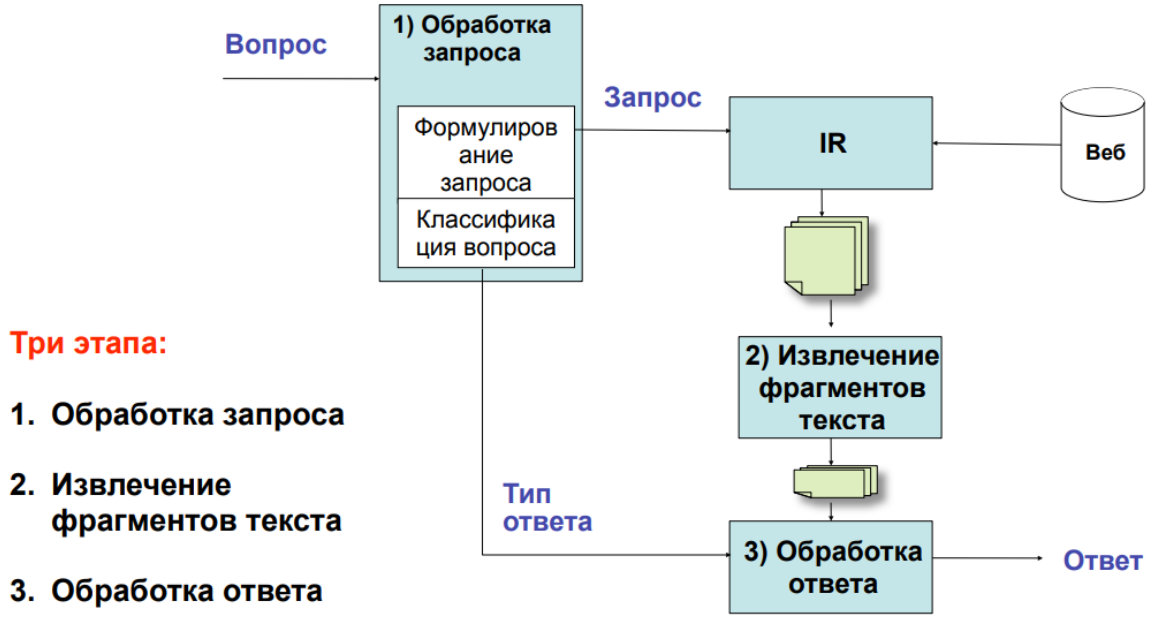


Рисунок 7.1 Обобщенная схема работы вопросно-ответной системы

Обработка ответа:

— Извлечение специфичного ответа из фрагмента
— Два основных класса алгоритмов
  – Основанные на шаблонах
  – Сбор ответа из N-грамм (N-gramm tiling) (рис 7.2)

Алгоритмы на основе шаблонов предполагают:

— Использование информации о типе в регулярных выражениях – Если тип ответа Млекопитающееся, извлечь именованные сущности Млекопитающееся из фрагмента
— Некоторые типы ответов (например, определения) не подразумевают конкретного типа именованной сущности в ответе – Использовать регулярные выражения (созданные вручную или автоматически)

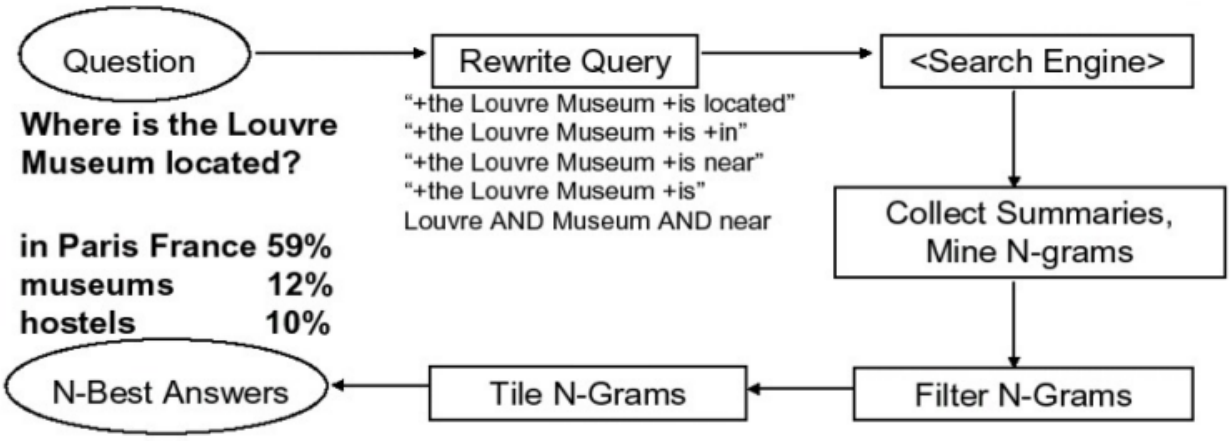


Рисунок 7.2 Обобщенная схема сбора ответа из N-грамм Архитектура AskMSR

Фильтрация и сбор ответа:

— Заново взвесить N-граммы с учетом типа ответа

— Собрать ответ

Аннотирование и реферирование

Извлечение короткого фрагмента текста - это адача автоматического реферирования

— Реферат состоит из частей оригинального текста

— Аннотация - главная мысль документа, сформулированная своими словами –Более компактная –Предполагает генерацию текста

Приложения

— Аннотации и рефераты к научным и другим статьям

— Реферированное новостей (несколько документов)

— Создание сниппетов

— Текст для мобильных устройств

— Реферат встречи • ..

2.3.2 Архитектура вопросно-ответной системы

Типичная обобщенная архитектура вопросно-ответной системы представлена на рисунке 7.3

Существует ряд различных подходов и принципов построения вопросно-ответных систем, основными из которых являются следующие:

1) метапоисковая система;

2) система поиска по аннотированному тексту;

3) экспертная система;

4) система поиска в коллекциях вопросов и ответов.

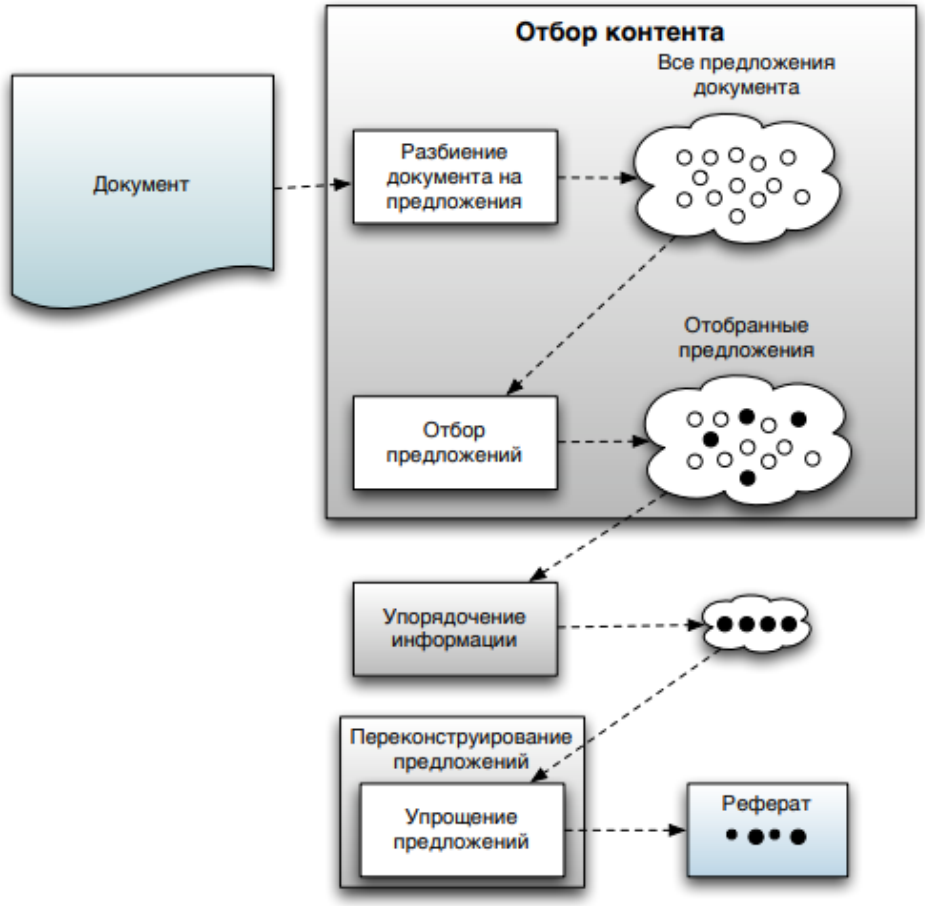


Рисунок 7.3 Обобщенная схема типичной архитектуры вопросно-ответной системы

Архитектура метапоисковой системы

Архитектура **метапоисковой** системы предусматривает использование существующей классической поисковой системы в качестве источника данных (рис. 1).

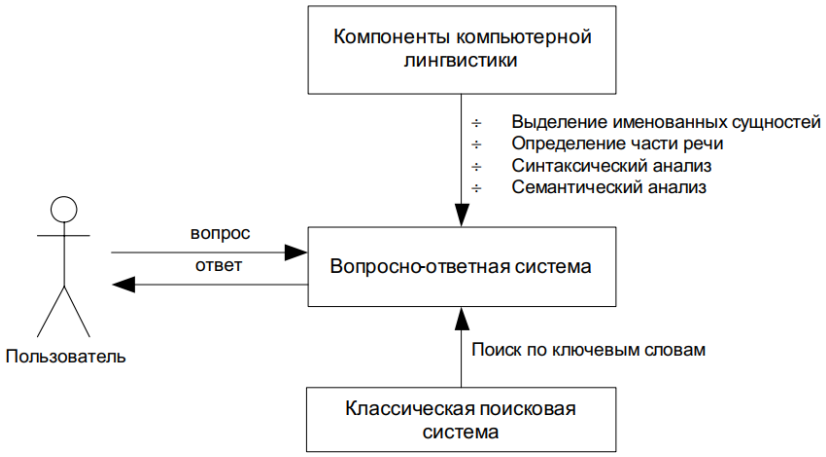


Рисунок 7.4 Обобщённая схема архитектуры метапоисковой вопросно-ответной системы

Вопросно-ответная система преобразует введённый пользователем вопрос на естественном языке в запрос в виде ключевых слов и анализирует

выдачу поисковой системы – несколько наиболее релевантных документов или их фрагментов (сниппетов).

Метапоисковые вопросно-ответные системы обычно формулируют запрос по ключевым словам на основе слов, входящих в опору вопроса. В англоязычных системах распространён приём расширения поискового запроса синонимами и гипонимами на основе лексической онтологии WordNet [4].

Результаты поиска по ключевым словам –сниппеты – обрабатываются существующими компонентами систем автоматической обработки текста (компьютерной лингвистики).

Преимущества метапоисковой архитектуры заключаются в следующем:

— отсутствие собственного индекса документов и, как следствие, отсутствие необходимости хранить огромный массив информации (для поиска в Интернете);
— гибкость, т.е. возможность абстрагироваться от задач поиска и компьютерной лингвистики. QA системы использует поисковую машину и лингвистические компоненты как чёрные ящики, и обычно не возникает проблем с заменой этих компонентов.

Метапоисковая QA система может использовать любые доступные инструменты для анализа сниппетов, независимо от математического аппарата, используемого в поисковой машине или лингвистических компонентах. Например, поисковая машина может работать на деревьях решений, построенных методами машинного обучения, синтаксический анализ может выполняться вероятностными методами, а QA система анализирует вопрос с помощью регулярных выражений и представлять сниппеты в виде графов синтаксических зависимостей.

Недостатками метапоисковой архитектуры являются:

— высокая вычислительная нагрузка в момент обработки вопроса, введённого пользователем, связанная с высокими вычислительными затратами на выполнение лингвистических задач;
— ограничения по управлению поиском (длина и «целостность» сниппетов, авторитетность источников и др.).

Принцип поисковых систем с коллекциями аннотированных документов

Вопросно-ответные системы, построенные по принципу поисковых систем с коллекциями аннотированных документов, имеют в своём составе поисковый индекс документов (рис. 7.5).

Данный индекс, в отличие от классических поисковых систем, дополняется специфическими для QA систем атрибутами. Построение индекса происходит с привлечением компьютерной лингвистики: каждый новый документ проходит автоматическую обработку текста на естественном языке, размечаются требуемые вопросно-ответные системы, построенные по принципу поисковых систем с коллекциями аннотированных документов объекты, затем они добавляются в индекс. Использование своего специального индекса позволяет преодолеть некоторые недостатки метапоисковой архитектуры.

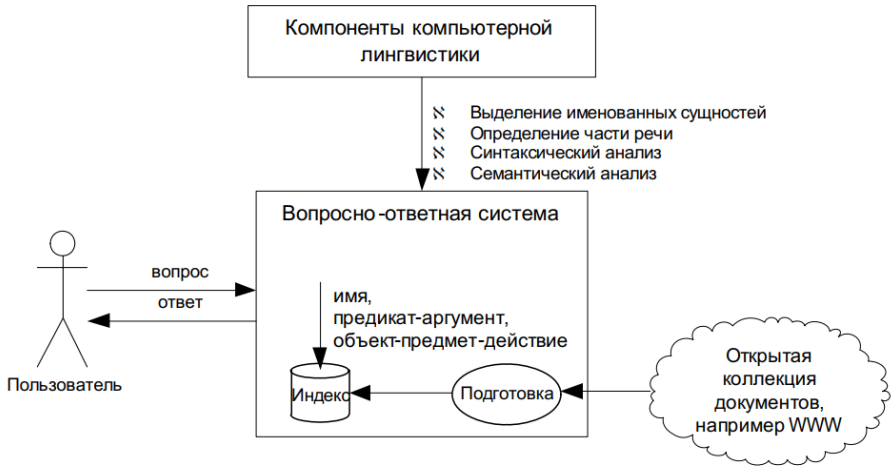


Рисунок 7.5 Обобщённая схема архитектуры вопросно-ответной системы, основанной на поиске по аннотированным текстам

Преимуществами поиска по аннотированному тексту являются:

— меньшая (по сравнению с метапоисковыми) вычислительная нагрузка в момент обработки вопроса пользователя в реальном времени благодаря специализированному индексу;

— возможность, благодаря специализированному индексу, организовать наиболее удобный поисковый аппарат.

Недостатки поиска по аннотированному тексту состоят в следующем:

— невысокая гибкость по сравнению с метапоисковой системой: на этапе построения индекса выбирается какая-то определённая модель представления текста;

Экспертная система

К классу экспертных систем можно отнести вопросно-ответные системы, построенные по принципу работы со структурированными базами данных (рис. 7.6).

Рисунок 7.6. Обобщённая схема архитектуры вопросно-ответной системы, построенной по принципу экспертных систем

Вопрос на естественном языке преобразуется в поисковый запрос к базе данных, содержащей структурированные факты, например, фреймы. В отличие от рассмотренных выше архитектур, такие системы могут выполнять логический вывод новой информации на основе множества разрозненных фактов. База данных фактов может быть построена автоматически в результате анализа коллекции документов. Процесс схож по построению аннотированного индекса, но он происходит на более детальном уровне обработки естественного текста, ведь извлекаются факты. Такие системы могут не хранить текст исходного документа, из которого был извлечён той или иной факт (фрейм).

Преимуществами QA, спроектированных на основе экспертной системы, являются:

— высокая скорость работы (по сравнению, например, с метапоисковой архитектурой);

— точность и достоверность результатов.

Недостатки заключаются в следующем:

— сильная зависимость от структуры фактов (фреймовой модели);

— необходимость выбирать только авторитетные исходные тексты для извлечения;

— вычислительная и организационная трудоёмкость построения базы фактов.

Социальный вопросно-ответный поиск

Другим подходом к автоматизации поиска ответов на вопросы пользователей является социальный вопросно-ответный поиск (*collaborative question answering*). Схематически архитектура такой системы представлена на рис 7.7

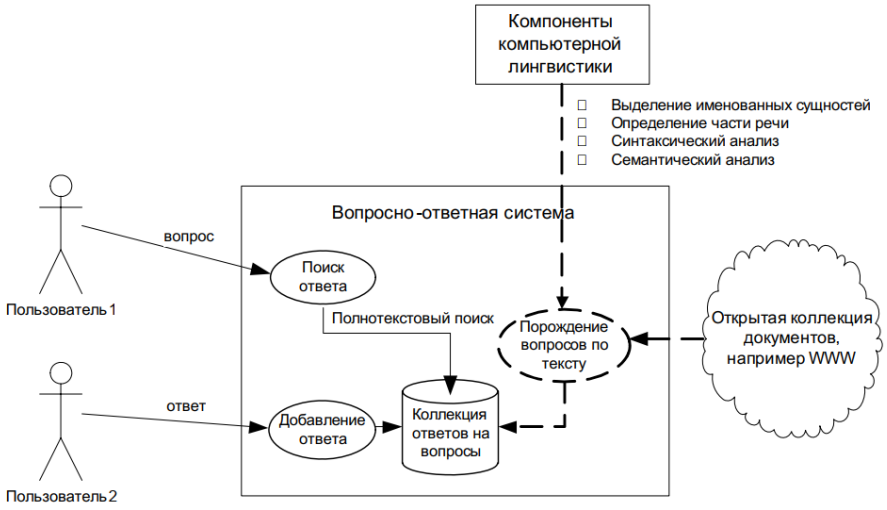


Рисунок 7.7 Обобщённая схема архитектуры системы социального вопросно-ответного поиска

В таких системах одни пользователи отвечают на вопросы других. Пользователь, имеющий информационную потребность, открывает страницу

веб-сайта системы и формулирует вопрос. Система сама находит похожие вопросы в коллекции вопросов и ответов и выдаёт нужный раздел. Если подобного вопроса не существует, создаётся новый раздел для обсуждения вопроса. Другие пользователи отвечают. Пользователь, задавший вопрос по мере появления ответов получает уведомления. Такие системы используют коллекции вопросов с ответами, которая может пополняться другими пользователями или автоматически.

Преимуществами использования коллекций вопросов и ответов являются:

— возможность развёрнутых, необязательно фактографических ответов;
— проверка достоверности ответов другими пользователями;
— низкие вычислительные затраты на поиск ответа в коллекции.

Недостатками являются:

— необходимость мотивации пользователей как для пополнения коллекции, так и для простановки оценок, особенно для ответов и вопросов, порождённых автоматически;
— трудоёмкость автоматического порождения коллекции, необходимость объёмного хранилища.

Кроме того, для QA имеется мощная архитектура **глубокого обучения** — Сеть Динамической Памяти (Dynamic Memory Network, далее — DNM). Обученная на тренировочном наборе из входных данных и вопросов, DNM формирует эпизодические вопросно-ответная системапоминания и использует их для генерации подходящих ответов.

Методы в извлекательном сокращении работают на основе выбора подмножества. Это достигается за счет извлечения фраз или предложений из статьи для формирования резюме. LexRank и TextRank — хорошо известные представители этого подхода, которые используют вариации алгоритм сортировки страниц Google PageRank. Это популярные алгоритмы машинного обучения без учителя на основе графо, соответственно классический и улучшеный.

Популярные фирменные решения QA:

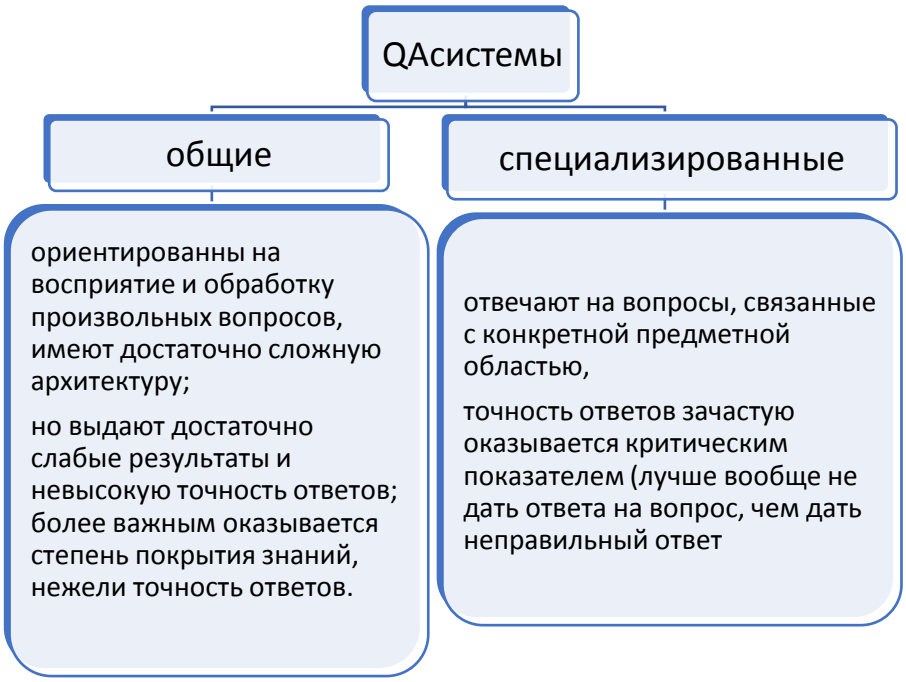
— Facebook Neural Attention — нейросетевая архитектура, которая использует локальную модель с механизвом внимания, способную генерировать каждое слово резюме в зависимости от входного предложения.

— Google Sequence-to-sequence придерживается кодер-декодер архитектуры. Кодер отвечает за чтение исходного документа и кодировку во внутреннее представление. Декодер отвечает за генерацию каждого слова в сводке на выходе и использует кодированное представление исходного документа.

— IBM Watson использует похожую Sequence-to-sequence модель, но со свойствами внимательной и двунаправленной рекуррентной нейросети.

2.3.3 Классификация вопросно-ответных систем

Классификация современных QA-систем схематически приведена на рисунке 7.8. Из схемы видно, что вопросно-ответные системы разделяют на 2 класса **общие** (open-domain) и **специализированные** (closed-domain).



Рисунке 7.8. Схема классификации современных QA-систем

Из схемы видно, что вопросно-ответные системы разделяют на 2 класса **общие** и **специализированные.** Системы класса общие могут основываться на архитектурах систем поиска по аннотированным текстам и социального поиска. Тогда как специализированные вопросно-ответные системы (closed-domain) чаще используют принципы мета поисковых или экспертных систем .

2.3.4 Производительность вопросно-ответной

Производительность вопросно-ответной системы зависит от эффективности используемых методов анализа текстов и от качества текстовой базы — если в ней нет ответов на вопросы, QA-система мало что сможет найти. Чем больше база — тем лучше, но только если она содержит нужную информацию. Большие хранилища (такие как Интернет) содержат много избыточной информации[2]. Это ведёт к следующим моментам:

Так как информация представлена в разных формах, то выше полнота информации. QA-система с большей вероятностью найдет ответ.

Правильная информация чаще повторяется, поэтому ошибки поиска ответов можно минимизировать.

Точность поиска информации существенно зависит от достоверности информации в хранилищах, а также от эффективности методов анализа информации и формирования ответов.

Направления развития вопросно-ответных систем

С момента появления первых прототипов вопросно-ответных систем их область применения значительно расширилась[4]. Например, их используют в ответах на вопросы, связанные со временем, геолокационные вопросы, вопросы определения понятий, библиографические, многоязыковые вопросы, вопросы, связанные с мультимедиа (визуальной, аудио- и видео- информацией). Изучаются смежные области, такие как построение интерактивных QA-систем (уточняющие вопросы, требующиеся для разъяснения первоначального), повторное использование ответов и представление знаний, использование логического вывода из имеющейся информации для получения ответов на вопросы и т. п., прогнозирование, какие вопросы могут быть заданы, анализ настроения.

2.3.5 QA-система Start

QA-система Start является примером общей вопросно-ответной системы, отвечающей на произвольные запросы, сформулированные на английском языке. Она разрабатывается в MIT Artificial Intelligence Laboratory под руководством Boris Katz. В Интернете эта система впервые появилась в 1993 и сейчас она доступна по адресу http://start.csail.mit.edu. При поиске ответа на вопрос система использует как локальную базу знаний, так и ряд информационных ресурсов в сети Интернет.

Система умеет отвечать на различные виды вопросов, которые условно можно разделить на следующие категории:

- Вопросы об определениях (What is a fractal?)
- Фактографические вопросы (Who invented the telegraph?)
- Вопросы об отношениях (What country is bigger, Russia or USA?)
- Списковые запросы (Show me some poems by Alexander Pushkin)

Ядром системы является База Знаний. Существуют 2 модуля: Парсер и Генератор, которые умеют, соответственно, преобразовывать тексты на английском языке в специальную форму (Т-выражения), в которой они сохраняются в Базе Знаний, и, наоборот, по набору Т-выражений генерировать англоязычные тексты.

2.3.6 Тернарные выражения

Тернарное выражение (Т-выражение) – это выражение вида <объект отношение субъект>. При этом в качестве объектов/субъектов одних Т-выражений могут выступать другие Т-выражения. Прилагательные, притяжательные местоимения, предлоги и другие части предложения используются для создания дополнительных Т-выражений. Остальные атрибуты предложения (артикли, времена глаголов, наречия, вспомогательные глаголы, знаки препинания и т.д.) хранятся в специальной структуре History, связанной с Т-выражением.

Например, предложение *“Bill surprised Hillary with his answer”* после прохождения через Парсер будет преобразовано в 2 тернарных выражения: *<<Bill surprise Hillary> with answer>* и *<answer related-to Bill>*. Информация о времени глагола surprise будет сохранена в структуре History.

Пусть системе, в Базе Знаний которой находятся 2 описанных выше Т-выражения, был задан вопрос: **“Whom did Bill surprise with his answer?”.** Обработка вопроса будет происходить в следующем порядке:

1. Анализатор Вопросов преобразует вопрос к виду шаблона, обратив инверсию, которая используется при формулировке вопросов в английском языке: **“Bill surprised *whom* with his answer?”**.
2. Парсер переводит предложение в 2 Т-выражения: **<<Bill surprise *whom*> with answer>** и **<answer related-to Bill>**
3. Полученный шаблон сверяется с Т-выражениями, находящимися в Базе Знаний. Найдено совпадение при ***Whom* = Hillary**
4. Генератор преобразует Т-выражения **<<Bill surprise Hillary> with answer>** и **<answer related-to Bill>** в предложение **“Bill surprised Hillary with his answer”** и выдаёт его в качестве ответа.

Аналогичным образом выполняется поиск ответов на вопросы вида “Did Bill surprise with his answer?”. Только в данном случае будет осуществляться поиск точного совпадения с выражениями в Базе, а не поиск по шаблону.

Таким образом, Т-выражения в некоторой степени сохраняют информацию о семантических связях между словами. В 2002 году были произведен ряд экспериментов с целью оценки эффективности организации поиска на основе Т-выражений по сравнению с поиском по ключевым словам. После обработки Парсером Энциклопедии с описаниями различных видов животных, системе был задан вопрос: “What do frogs eat?” («Что едят лягушки?»). Описанный выше метод поиска выдал 6 ответов, из которых 3 были правильными. Поиск на основе ключевых слов по исходным документам выдал 33 результата, среди которых были те же 3 правильных ответа, но, кроме того, встречались случайные совпадения слов **frogs** и **eat** (например, ответы на вопрос “Кто ест лягушек?”). Таким образом, поиск на основе Т-выражений выдал в 10 раз меньше неверных ответов.

2.3.7 S-правила

Помимо Т-выражений, в Базе Знаний также хранится перечень S-правил. Это правила перевода Т-выражений в эквивалентные формы. Дело в том, что одну и ту же идею в естественном языке можно выразить различными способами. Например, предложения *“Bill’s answer surprised Hillary”* и *“Bill surprised Hillary with his answer”* эквивалентны. Но Т-выражения, получаемые при проходе этих предложений через Парсер различны: *<answer surprise Hillary>*, *<answer related-to Bill>* и *<<Bill*

surprise Hillary> with answer>, <answer related-to Bill>. Поэтому вводится S-правило ***Surprise***:

<<n_1 surprise n_2> with n_3>, <n_3 related-to n_1> = <n_3 surprise n_2>, <n_3 related-to n_1>

Where n_i € *Nouns*

С помощью таких правил можно описать так называемые лингвистические вариации, то есть эквивалентные трансформации языковых конструкций:

- Лексические (синонимы)
- Морфологические (однокоренные слова)
- Синтаксические (инверсии, активный/пассивный залог, …)

Кроме того, S-правила могут описывать логические импликации. Например:

<<A sell *B>* to *C>* = *<<C* buy *B>* from *A>*

2.3.8 Лексикон

Многие S-правила применимы к группам слов. Например, описанное ранее S-правило **Surprise** выполняется не только для глагола surprise, но также для любого глагола из так называемой группы эмоционально-реакционных глаголов. Для того, чтобы не плодить S-правила был создан Лексикон, в котором хранятся все слова английского языка. С каждым словом связан перечень групп, к которым оно относится. Теперь S-правило Surprise можно сделать ещё более абстрактным:

<<n_1 v n_2> with n_3>, <n_3 related-to n_1> = <n_3 v n_2>, <n_3 related-to n_1>,

Where n_i € Nouns, v € emotional-reaction-verbs

2.3.9 WordNet

Кроме Лексикона, в котором хранятся сгруппированные по различным синтаксическим и семантическим признакам слова, система Start использует ещё один мощнейший инструмент обработки семантики слов – словарь **WordNet**. В качестве базовой единицы в этом словаре используется понятие **синсета.**

Синсет – это некоторый смысл, значение. Различные слова могут иметь одно и то же значение (синонимы), поэтому относиться к одному синсету, и, наоборот, одно слово может иметь несколько значений, то есть принадлежать нескольким синсетам. Кроме того, в словаре WordNet введены отношения между синсетами.

Например, между существительными существуют следующие отношения:

— Гиперонимы: Y – гипероним X, если X – разновидность Y (фрукт – гипероним персика)
— Гипонимы: Y – гипоним X, если Y – разновидность X (персик – гипоним фрукта)

— Равные по рангу: X и Y равны по рангу, если у них общий гипероним (персик и яблоко – равные по рангу)
— Голонимы: Y – голоним X, если X – часть Y (персик – голоним косточки)
— Меронимы: Y – мероним X, если Y – часть X (кожура – мероним персика)

Таким образом, в словаре WordNet описаны отношения между смыслами вида общее-частное и часть.

WordNet используется при поиске совпадений в Базе Знаний. Например, если в Базе хранится T-выражение <*bird can fly*> и в словаре WordNet определено, что canary – гипоним bird. Пусть был задан вопрос *"Can canary fly?"*. Парсер преобразует этот вопрос в выражение <canary can fly>. Не найдя совпадений в Базе, Start применит WordNet и попробует найти ответ на более общий вопрос: *"Can bird fly?"*. На этот вопрос будет найден ответ *Yes*, из чего, учитывая, что canary – разновидность bird Start сделает вывод о том, что *"canary can fly"*.

2.3.10 Omnibase

Для поиска ответов на фактографические вопросы типа "When did Beethoven die?" или "What is the capital of England?" Start использует базу **Omnibase**. В этой базе используется иная модель хранения информации: «объект-свойство-значение». Например информация "Federico Fellini is a director of La Strada" сохранится в базе Omnibase в виде La Strada – director – Federico Fellini. Здесь La Strada – объект, director – свойство, а Federico Fellini – значение этого свойства. При такой модели данных поиск необходимой информации происходит достаточно быстро и эффективно.

Для поиска информации Omnibase использует большое количество внешних источников данных из сети Интернет: Wikipedia, Google, Internet Movie Database и т.д. При этом извлечение данных из внешнего источника происходит через так называемую обертку (wrapper) – модуль, обеспечивающий доступ к внешней базе через запросы вида «объект-свойство». Для определения источника, в котором хранится информация о том или ином объекте, Omnibase использует Каталог Объектов, в котором каждому объекту сопоставлен источник данных. Например, объекту La Strada соответствует база imdb-movie (Internat Movie Database). Определив базу, в которой следует искать, Omnibase посылает запрос к обертке этой базы: (La Strada, director) и получает ответ Frederico Fellini.

2.3.11 Популярные QA-системы и демоверсии

— Одна из первых размещённых в интернет вопросно-ответная система START на сайте MIT.
— Вопросно-ответная система AskNet Search на сайте asknet.ru (первоначально Stocona Search).
— Вопросно-ответная система BrainBoost на сайте Answers.com (англ.)рус. (первоначально BrainBoost.com).
— QA-система, встроенная в поисковик Ask.com.
— Вопросно-ответная система OpenEphyra с открытым исходным кодом.

— Проект Evi от True Knowledge (англ.)рус..
— Специализированные QA-системы
— EAGLi: MEDLINE question answering engine (англ.).

2.3.12Вопросы к лекции№7

1. Дайте определение понятия вопросно-ответной системы.
2. Поясните механизм работы вопросно-ответной системы .
3. Расскажите о назначении вопросно-ответных систем.
4. Каковы основные функции архитектуры вопросно-ответной системы и каково ее применение в информационном поиске?
5. Перечислите широко используемые методы установление смысла вопроса и порождение ответов в информационном поиске и обработке естественного языка.
6.
7. Поизведите классификацию вопросно-ответных систем и коротко охарактеризуйте каждый класс.
8. Каковы особенности общих QA-систем.
9. Каковы особенности специализированных QA-систем.
10. Расскажите о работе QA-системы Start.
11. Дайте определение понятия трернарное выражение.
12. Каков смысл использования S-правил?
13. Для каких преобразований используется Лексикон?
14. Каково назначение WordNet?
15. Расскажите об использовании базы Omnibase.
16. Для чего необходимо внедрение аннотаций?
17. Назовите и коротко охарактеризуйте популярные QA-системы.

2.4 Лекция № 8 Программирование и проектирование систем обработки естественных языков: задачи морфологического анализа, морфологический разбор, стемминг, лемматизация.

План лекции №8

1. Обобщенный алгоритм обработки текста на естественном языке.
2. Морфологический и синтаксический разбор текстов.
3. Фазы обобщенного алгоритма обработки текста на естественном языке: синтаксический анализ, семантический анализ, прагматический анализ
4. Анализ и индексирование текста.
5. Стемминг.
6. Лемматизация.
7. Сравнение функций стемминга и лемматизации для поиска текста с учетом морфологии.

2.4.1 Обобщенный алгоритм обработки текста на естественном языке

Обобщенный алгоритм обработки текста на естественном языке представлен схематически на рис. 8.1.

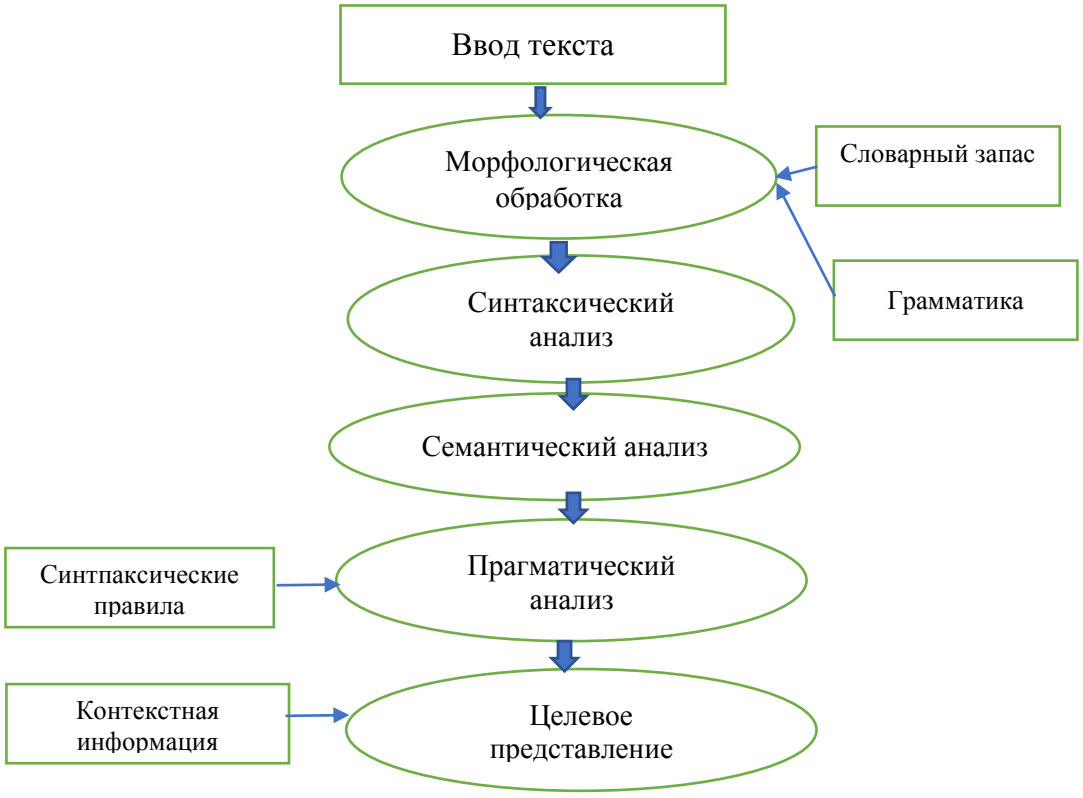


Рисунок 8.1 — Схема обобщенного алгоритма обработки текста на естественном языке

2.4.2 Корпус текстов

В лингвистике ко́рпус текста – совокупность текстов, подобранная и обработанная определенным образом для использования в качестве базы для исследования языка. Корпус применяют для анализа и проверки статистических

гипотез, подтверждения лингвистических правил в данном языке и других исследований.

Главные свойства корпуса:

– электронный,
– репрезентативный,
– размеченный,
– прагматически ориентированный.

Выделяют классы корпусов - одноязычные, двуязычные и многоязычные.

В свою очередь многоязычные и двуязычные делятся на два типа:

— параллельные — множество текстов и их переводов на один или несколько языков;
— сопоставимые (псевдопараллельные) — оригинальные тексты на двух или нескольких языках.

Разметка корпуса заключается в приписывании текстам и их компонентам специальных тегов: лингвистических и внешних. Различают типы разметки: морфологическая, семантическая, синтаксическая, анафорическая, просодическая, дискурсная и т. д.

Корпус должен быть конечным по размеру, для этого нужно выполнить выборку и пропорционально включить широкий спектр типов текста, чтобы обеспечить хороший **дизайн** корпуса.

Репрезентативность или **представительность корпуса** является определяющей чертой его дизайна. Известны определения репрезентативности корпуса двух великих исследователей — Лича и Бибера.

Leech (1991): «корпус считается представителем языкового разнообразия, которое он должен представлять, если результаты, основанные на его содержании, могут быть обобщены до указанного языкового разнообразия».

Biber (1993):«репрезентативность относится к степени, в которой выборка включает полный диапазон изменчивопросно-ответная системати в популяции».

Репрезентативность корпуса характеризуется:

— баланс,
— выборка.

Баланс корпуса — диапазон жанра, включенного в корпус. Научной меры для баланса нет, его оценка скорее интуитивна, т.е., можно сказать, что принятый баланс определяется только его предполагаемым использованием.

При получении репрезентативной **выборки** учитывается механизм отбора данных.

Размер корпуса зависит от цели использования, вида запроса, методики поиска и источника данных.

В 60-70-е годы 20 века размер корпуса был, к примеру, ориентировочно равен 1 миллион слов (коричневый и LOB), то к 90годам100 миллионов слов (британский национальный корпус) и к началу 21века 650 миллионов слов (банк английского корпуса)

Применение корпусов

Применение корпусов не ограничивается рамками , обработки естественного языка и компьютерной лингвистики.

Они широко используются в технологии распознавании речи и машинном переводе, при расшифровке древний письменности Одним из самых коротких по времени корпусов могут быть тексты писем Амарны за 15–30 лет (1350 г. до н.э.). Корпус древнего города, (например, «Кюльтепа тексты» из Турции), может пройти через серию корпусов, определенную дата их находку сайта.

2.4.3 Морфологический разбор текста

Морфологический и синтаксический разбор текстов в поисковой системе это конечный автомат. Это семантическая нейронная сеть, имеющая структуру синхронизированного линейного дерева.

Причем метод извлечения знаний из текста на естественном языке и построение ответа производится при помощи семантической сети. Эту сеть можно представить в виде семантического дерева.

Для понимания вычислительной системой смысла текста необходимо решить разбора. В разбор текста включены несколько последовательных операций: морфологический, синтаксический и семантический анализ. Порядок следования этих операций важен, в связи с тем, что определение всех синтаксических признаков слов, необходим для анализа семантики. Для решения задач анализа текста применяют семантическую нейронную сеть, близкую по свойствам формальной нейронной сети Маккаллока-Питтса.

Извлечения смысла из текста можно представить в виде подсети, при этом отдельный нейрон инициирует элементарное понятие, в соответствие с этапом обработки, к которому относится данный подслой нейронной сети. Понятия естественного языка с законченным смыслом, такие как символ, слог, слово, предложение, абзац, весь текст можно отнести к элементарным понятиям [29], которые структурированы по определенным уровням: символ, слог, слово, словосочетание.

2.4.4 Синтаксический анализ, семантический анализ, прагматический анализ

Вернемся к фазам обобщенного алгоритма обработки текста на естественном языке представленном на схеме в начале лекции (рис. 8.1)

Нами была изучена первая фаза НЛП - **морфологическая обработка.** Вспомним, что ее цель - разбиение фрагментов ввода языка на наборы токенов, соответствующих абзацам, предложениям и словам. Аббревиатура НЛП произошла от NLP - Natural Language Processing и общепринята. НЛП транслитеризация NLP часто встречается ее в литературе и означает «обработка текстов на естественном языке».

Следующий этап обработки текстов на естественном языке **синтаксический анализ** .На этом этапе следует выяснить, правильность формирования предложений и произвести разбиение предложений на

структуры, которые показывают синтаксические отношения между различными словами. Например, синтаксический анализатор отклонит предложение типа *«Университет идет к студенту»*.

Далее производится **семантический анализ** текста, который позволит нарисовать точное значение, проверит текст на осмысленность. Например, предложение *«Ледяной жар»* будет отклонено на этом этапе как бессмысленное.

Прагматический анализ сопоставляет фактические объекты или события, которые имеются в данном контексте, с ссылками, полученными на этапе семантического анализа. Например, предложение *«Сложите игрушки в коробку на этажерке»* может интерпретировано двумя вариантами, и прагматический анализатор должен сориентироваться между ними.

2.4.5 Анализ и индексирование текста

Нормализатор – блок индексатора, структурирующий документ для удобства поиска

Чтобы неструктурированную информацию преобразовать в структурированную необходимо выяснит, в каких документах содержится запрашиваемое слово. Это можно выполнить так:

- проверить все тексты документов:
- составить список документов, в которых встречается слово

Для поиска по текстам используется инвертированный индекс. Известно, инвертированный индекс для каждого слова коллекции документов содержит списки документов, в которых оно встретилось. Инвертированного индекс может содержать дополнительно позицию слова в каждом документе.

Вхождение слова в документы описывается id документов, TF-IDF, бинарный фактор: «попало слово в заголовок или не попало», и др. Это важно для ранжирования. Индекс может строиться по леммам. Стоп-слова обычно исключаются. Для ускорения вычисления пересечений используют эвристику skip-pointer. В случае запроса, состоящего из большого количества слов, используют функцию кворума, которая разрешает проход на следующую стадию ранжирования часть документов, в которых встретились не все слова из запроса.

2.4.6 Стемминг

Стемминг (stemming)- отсечение от слова окончаний и суффиксов, чтобы оставшаяся часть, называемая stem, была одинаковой для всех грамматических форм слова. Разумеется в таком виде стеммер может работать только с языками, которые реализуют словоизменение через аффиксы. Примерами таких языков являются русский и английский.

Обычно стеммером пользуются для поиска текста с имитацией учета морфологии. Под имитацией подразумевается неустранимо большое количество ошибок и нерелевантных результатов, которые возникают, если применять только стеммер. В русском языке источником ошибок при

стемминге являются всевозможные изменения корня слова - беглые гласные, к примеру. Наглядно проблемы, связанные с использованием стеммера, можно продемонстрировать для русского существительного кошка. Родительный падеж множественного числа имеет форму кошек. Таким образом, самый длинный общий префикс всех форм существительного кошка - это кош. Если выполнить поиск текста по этому префиксу, то в результатах с большой вероятностью будут такие слова, как кошмар. Замечу, что обычно реализации стеммера идут немного другим путем и допускают ошибку иного рода - они возвращают при стемминге префикс кошк и таким образом из результатов поиска исчезают фрагменты текста с формой кошек.

В качестве решения проблемы плохих результатов поиска со стеммером для русского языка можно использовать два дополнительных модуля грамматического словаря - лемматизатор и флексер (склонение и спряжение). С помощью лемматизатора можно приводить слова к базовой форме, поэтому после сопоставления слова со стемом можно уточнить результат с помощью лемматизации. Второй модуль - флексер, который умеет выдавать все грамматические формы слова на основе базовой. Это позволяет уточнять результаты поиска, проверяя найденные фрагменты по набору форм ключевого слова.

Допускаемые при стемминге ошибки можно классифицировать следующим образом.

Ошибки стемминга 1го рода - стем дает слишком большое обобщение и поэтому сопоставляется с грамматическими формами более чем одной словарной статьи. Это самая многочисленная группа ошибок стемминга. К примеру, если при стемминг вами даст вам, то в дальнейшем поиск текста даст совпадение с вампир . Об этом говорит сайт https://intellect.icu . В русском языке мжет быть весьма трудно полностью устранить данные ошибки, к примеру глагол пасть при спряжении дает формы пади и пал, в результате стемминг дает па, и это очень большое расширение при поиске. Впрочем, ошибки такого типа могут рассматриваться и как способ включить в поиск однокрненные слова - в примере с кошкой это могут быть формы прилагательного кошачий. Компенсация ошибок первого рода успешно выполняется либо введением списка стоп-слов, либо более качественно - лемматизатором или флексером.

Ошибки стемминга 2го рода - усечение формы дает слишком длинный стем, которые не сопоставляется с некоторыми грамматическими формами этого же слова. К таким ошибкам приводит стремление разработчика стеммера найти компромисс с ошибками 1го рода в случае, когда при словоизменении меняется основа слова. Такие слова есть даже в крайне регулярном в плане словоизменения английском языке, например - группа неправильных глаголов. В русском случаи изменения основы даже не являются основанием для отнесения слова к группе неправильных, настолько часто это явление. В качестве примера, на котором обычно спотыкаются многие реализации стеммера, можно взять слова кошка и пачка, которые имеют формы кошек и пачек. Обычно стеммеры выполняют в этих случаях усечение до кошек и

птичек, которые несопоставимы с формами родительного и винительного падежа множественного числа.

Ошибки стемминга 3го рода - стем построить невозможно из-за изменения в корне слова, которое оставляет единственную букву в стеме. Либо модель словоизменения подразумевает использование приставок. Пример для первого случая - глагол впиться, имеющий форму вопьемся. Второй случай возникает в рамках грамматического словаря для сравнительной степени прилагательных и наречий в русском языке - например покрасивее как форма прилагательного красивый, или помедленнее как форма наречия медленно.

Стеммер - это упрощенный алгоритм морфологического разбора слова, оптимизированный под максимально быстрое нахождение префикса, общего для всех грамматических форм заданного слова. Обычно получамая при стемминге основа включает в себя морфологический корень, вместе с приставкой. У стеммера всегда есть некоторый процент ошибок, проистекающих из особенностей словоизменения естественного языка и невозможности согласовать примитивную идею отсечения "окончаний" со русским словоизменением, а тем более с такими языковыми явлениями, как беглые гласные. В русском лексиконе примерно 1.7% словоформ имеют такую особенность, например мешок-мешки, взять-возьму. Даже такой регулярный язык, как английский, имеет обширный набор исключений из регулярных правил - неправильные глаголы исуществительные, склоняющиеся не по общему правилу.

2.4.7 Лемматизация

Лемматизатор - Разбор слова по составу: как в школе по частям слова(приставка,корень,суффикс, окончание), только автоматически (рис.8.2)

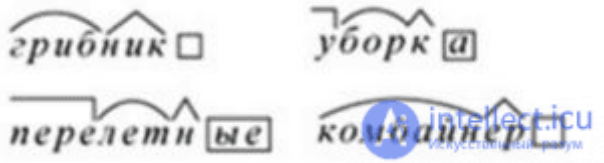


Рисунок 8.2 — Разбор слова по составу

Крупнейший лемматизатор русского языка: AOT.ru (py_morphy, PHPMorphy, …)

Плюсы: получение термина в нормально форме, вопросно-ответная системапринимаемой человеком

Минусы: долго работает

Таблицы лемматизатора

В составе схемы SQL словаря есть отдельные таблицы, разработанные исключительно для выполнения лемматизации:

LEXEMES_1 пары слово-лемма, а также дополнительное поле - количество альтернативных нормальных форм.

LEXEMES_N пары слово-лемма, в отличие от предыдущей таблицы может быть несколько записей для каждого слова.

LEMMAS - вспомогательная таблица, список нормальных форм, на которые ссылаются LEXEMES_1 и LEXEMES_N.

С помощью такого запроса к базе данных

SELECT lemma_**COUNT**, **COUNT**(*)
FROM lexemes_1
GROUP BY lemma_**COUNT**

Можно определить, насколько велик в естественном языке процент слов, которые дают неоднозначную лемматизацию, например для русского языка, примерно 1% слов русского языка могут доставить некоторые трудности при приведении к нормальной грамматической форме.

К примеру, такой запрос к базе данных словаря

SELECT lexeme, lemma
FROM lexemes_n, lemmas
WHERE lemmas.id=id_lemma **AND** lexemes_n.lexeme='*РОЙ*'

даст такие варианты лемматизации(рис. 8.3)

lexeme	lemma
РОЙ	РЫТЬ
РОЙ	РОЙ

Рисунок 8.3 — Варианты лемматизации слова *«рой»*

Повелительная форма глагола *рыть* и две падежные формы существительного *рой* текстуально совпадают.

Выбор между быстрой, но иногда неполной лемматизацией, и медленной и исчерпывающей, зависит от решаемой задачи.

Например, в поисковой системе скорость индексирования играет решающую роль, так как лемматизация миллионов лексем становится основным занятием процессора. Поэтому индексатор с командой -index wordforms использует упрощенную лемматизацию - берет первую нормальную форму, игнорируя остальные варианты.

С другой стороны машинный переводчик должен максимально аккуратно учитывать все возможные нормальные формы переводимых слов, поэтому скорость лемматизации приносится в жертву точности.

Процедуры API Грамматического Словаря для лемматизации

В API есть несколько вызовов, которые позволяют прямо или с дополнительными шагами выполнять приведение слова к базовой форме - лемматизацию. Большинство из них одинаково успешно работают с любым из поддерживаемых в проекте языков, разумеется при наличии соответствующего словаря.

Основные процедуры для получения базовой грамматической формы

Два основных вызова, разработанных именно как лемматизаторы:

sol_TranslateToBase

Упрощенный, но самый быстрый вариант. Ищет единственную нормальную форму слова. Если слово может быть приведено к нескольким

базовым формам, например для *мою* могут быть варианты *мыть* и *мой*, то используется одна любая из альтернатив на усмотрение лемматизатора.

sol_TranslateToBases

Более сложный и медленный вариант предыдущей процедуры. Если поданное на вход слово можно лемматизировать более чем единственным способом, то возвращает все варианты.

Пример использования обеих процедур можно найти в исходных текстах демо-программы **...\demo\ai\solarix\Grammar_Engine\Lemmatizer_Russian**.

Прочие процедуры для получения леммы

Далее описаны другие процедуры API, которые решают эту же или близкую задачу.

sol_SeekWord

Это быстрый поиск id словарной статьи по любой грамматической форме. Как и sol_TranslateToBase, берется первый вариант базовой формы, если есть альтернативы. Возвращается целое число - ключ словарной статьи, которое можно использовать для таким операций, как получение названия статьи, склонения и спряжения, определения грамматических свойств слова.

sol_FindWord

Медленный вариант поиск словарной статьи по любой грамматической форме.

sol_ProjectWord

Поиск всех возможных словарных статей, чьей грамматической формой является слово. Процедура вернет список, элементы которого позволяют получить целочисленный id подходящих словарных статей. Особенность данной процедуры в том, что ее алгоритм может для неизвестного слова попытаться подобрать статью и приставку-суффикс, чтобы найти статью-оригинал. Например, для *суперкот* будет найдена словарная статья *кот*, так как продуктивная приставка *супер*- включена в число проверяемых.

sol_ProjectMisspelledWord

Расширенный вариант предыдущей процедуры, позволяет искать словарные статьи по форме с возможными опечатками - пропусками букв, заменами, лишними буквами.

В некоторых приложениях бывает полезно не просто лемматизировать слово, а получать по возможности существительное, от которого получено слово, например *страшный* - *страх*. В составе API есть соответствующий вызов:

sol_TranslateToNoun

Выполнение лемматизации с SQL словарем

В схеме SQL словаря есть несколько таблиц, которые можно создавать и использовать автономно, позволяющих выполнять поиск базовых форм слов.

Для русского языка, как уже отмечалось в описании процедуры лемматизации через API грамматического словаря, главная проблема - наличие неоднозначностей.

С помощью несложного запроса можно увидеть, что всего в русском лексиконе имеется чуть больше 400 пар существительное-глагол, у которых хотя бы одна грамматическая форма омонимична.

Этот запрос к базе данных словаря:

```
SELECT DISTINCT E1.name AS "сущ", E2.name AS "глаг", F2.name AS "форма"
 FROM  sg_form F1, sg_form F2, sg_entry E1, sg_entry E2
 WHERE E1.id_class=8 AND -- русские существительные
    F1.id_entry=E1.id AND -- их грамматические формы
    F2.name=F1.name AND -- формы совпадают
    E2.id=F2.id_entry  AND E2.id_class IN (13,14) -- русские глаголы и инфинитивы
```

В результате получим примерно такую таблицу (Таблица 8.1 показано несколько записей):

Таблица 8.1 Результат лемматизации

сущ	глаг	форма
явь	явить	яви
штурман	штурмануть	штурману
шлем	слать	шлем
шило	шить	шило
шило	шить	шила
шило	шить	шил
шея	шить	шей
шаль	шалить	шали
чертенок	чертить	чертят

Теперь рассмотрим, как получить базовую форму слова по любой грамматической.

Запрос к базе словаря имеет вид:

```
SELECT DISTINCT Lower(sg_class.name), sg_entry.name
 FROM  sg_form, sg_entry, sg_class
```

```
WHERE sg_form.name='пни' AND
    sg_entry.id=sg_form.id_entry AND
    sg_class.id=sg_entry.id_class -- покажем часть речи
```

Обратите внимание, что мы никак не фиксируем язык обрабатываемых слов.

Результат вышеприведенного запроса, в котором ищутся базовые формы для слова *пни*, представлен на рисунке 8.4

глагол	пнуть
существительное	пень

Рисунок 8.4 — Результат выполнения запроса, в котором ищутся базовые формы для слова *пни*

Выполнение лемматизации в случае многоязычного словаря

Как уже отмечалось выше, база данных правил лемматизации генерируется автоматически компилятором словаря на основе загруженных в лексикон словарных статей и их грамматических форм. Генератор построен с максимально широкими допущениями на лексический состав словаря, поэтому вполне реально получить модуль, возвращающий лемму и для русских, и для английских слов.

Лемматизация и субстантивация

Наличие тезауруса, ориентированного на машинную обработку текста, позволяет реализовывать более сложные преобразования слов. К примеру, вместо простой лемматизации можно одним запросом возвращать базовую форму существительного, к которому приводится словарная статья. Например, прилагательное *кошачий* является дериватом от существительного *кошка*. Можно приводить не только к форме существительного, но и к форме инфинитива, например *причастие* бегающий приводится к инфинитиву *бегать*. Более того, инфинитив *бегать* связан с существительным *бег*. Все эти связи, образующие сложную семантическую сеть, доступны для SQL запросов, пример одного из которых показан далее.

Итак, мы имеем несколько слов, принадлежащих к разным частям речи. Нам нужно для каждого из них выполнить лемматизацию, а затем по возможности найти форму существительного. Обратите внимание, что результат лемматизации может не допускать приведения к существительному - в этом случае мы вернем простую лемматизацию.

В справочнике типов связей можно найти числовой код нужного нам типа **в_сущ=39**.

Составляем запрос к базе словаря:

```
SELECT DISTINCT COALESCE( E2.name, E1.name )
 FROM  sg_form, sg_entry E1 LEFT JOIN sg_link ON id_entry1=E1.id
```

```
                AND istate=39 -- тип связи <в_сущ>
            LEFT JOIN sg_entry E2 ON E2.id=id_entry2
WHERE sg_form.name IN ( 'сияющими', 'звенело', 'по-кошачьи' )
    AND E1.id=sg_form.id_entry
```

Результат его исполнения вышеуказанного запроса представлен на рисунке 8.5

Рисунок 8.5 — Результат выполнения запроса к базе словаря

Параметры лемматизатора

Лемматизатор может использоваться как автономно, так и загружаться в составе грамматического словаря. Параметры подключения и работы лемматизатора в этих случаях задаются в файле конфигурации словаряв отдельном XML-узле - см. здесь.

Дополнительные алгоритмы лемматизации текста

Кроме специализированного лемматизатора, описанного выше, нормализация текста может быть выполнена другими способами. Отметим из них два самых интересных.

Морфологический анализатор неявно дает на выходе информацию о леммах, так как он распознает для каждого исходного слова принадлежность к определенной словарной статье. У словарной статьи можно получить начальную форму слова, и таким образом решить задачу. Побочным эффектом такого алгоритма будет учет контекста слова и отбрасывание альтернативных вариантов распознавания.

Можно пойти дальше, и учесть информацию из тезауруса. Взяв результаты морфологического анализа предложения, можно для каждого слова проверить, не является ли оно уменьшительной (*котик*) или усилительной (*котище*) формой слова, и привести к нейтральной (*кот*). Более того, можно вопросно-ответная системапользоваться информацией о дериватах, к примеру выполнить приведение глаголов, деепричастий и причастий к форме абстрактного существительного движения или действия: *изменять-изменение*.

На рисунках 8.6-8.8 отображены виды лемматизации текста для одного и того же предложения.

Рисунок 8.6 — Результат выполнения простой лемматизации

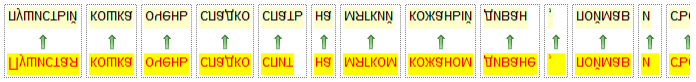


Рисунок 8.7 — Результат морфологического разбора предложения

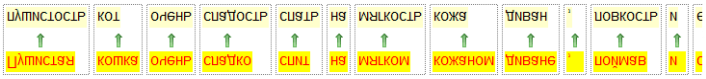


Рисунок 8.8 — Результат глубокой нормализации текста с приведением к существительным с помощью тезауруса

Стоп-слова

Стоп-слово – слово, которое встречается во многих текстах, но никак текст не характеризует

Примеры: союзы, междометия

2.4.8 Вопросы к лекции№8

1. Отобразите и поясните суть обобщенного алгоритма обработки текста на естественном языке.
2. Как производится морфологический и синтаксический разбор текстов?
3. Как произвести анализ и индексирование текста?
4. Поясните суть процедуры «стемминг».
5. Поясните суть процедуры «лемматизация».
6. Произведите сравнение функций стемминга и лемматизации для поиска текста с учетом морфологии.
7. Опишите фазы обобщенного алгоритма обработки текста на естественном языке: синтаксический анализ, семантический анализ, прагматический анализ

2.5 Лекция № 9 Программирование и проектирование систем обработки естественных языков: минимальное расстояние редактирования, алгоритм подсчета расстояния Левенштейна.

2.5.1 Программирование и проектирование систем обработки естественных языков (продолжение)

План лекции

1. Структура и принцип действия семантической нейронной сети, выполняющей морфологический и синтаксический разбор как синхронизированное линейное дерево
2. НЛП — Лингвистические ресурсы
3. Обработка естественного языка — Python.
4. Минимальное расстояние редактирования, алгоритм подсчета расстояния Левенштейна

2.5.2 Структура и принцип действия семантической нейронной сети, выполняющей морфологический и синтаксический разбор как синхронизированное линейное дерево

Рассмотрим **схему семантической нейронной сети, выполняющей морфологический и синтаксический разбор** как синхронизированное линейное дерево:

— Линейное дерево многослойно. 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.

Принцип действия семантической нейронной сети, выполняющей морфологический и синтаксический разбор как синхронизированное линейное дерево буде таков:

— 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

2.5.3 НЛП — Лингвистические ресурсы

Обработка текста производится его представлением в виде лингвистического ресурса их изучение мы начали в рамках предыдущей лекции.

Корпус — это большой и структурированный набор машиночитаемых текстов, которые были созданы в естественной коммуникативной обстановке.

Рассмотрим несколько примеров корпусов.

TreeBank Corpus [7,14] - лингвистически проанализированный текстовый корпус, который аннотирует синтаксическую или семантическую структуру предложения. Джеффри Лич ввел термин «древовидный банк», который означает, что наиболее распространенным способом представления грамматического анализа является древовидная структура. Как правило, древовидные банки создаются в верхней части корпуса, который уже снабжен тегами части речи [7,14].

Типы TreeBank Имеется 2 наиболее распространенные типы коруса: семантические и синтаксические древовидные банки [7,14].

Семантические древовидные - используют формальное представление семантической структуры предложения. Они различаются по глубине своего семантического представления. Команды роботов Treebank, Geoquery, Groningen Meaning Bank, RoboCup Corpus — вот некоторые примеры семантических древовидных банков.

Синтаксические древовидные - входные данные для систем синтаксического древовидного банка являются выражениями формального языка, полученного в результате преобразования проанализированных данных древовидного банка. Выходы таких систем основаны на предикатном логическом значении представления.

К настоящему времени созданы различные синтаксические древовидные блоки на разных языках. Например, **Penn Arabic Treebank, Columbia Arabic Treebank** — это синтаксические Treebank, созданные на **арабском** языке. **Sininca** синтаксический Treebank создан на китайском языке. **Люси, Сьюзен** и **BLLIP WSJ** создали синтаксический корпус на английском языке.

Приложения TreeBank Corpus[7,14].

Для разработки современных систем обработки естественного языка, таких как метки части речи, парсеры, семантические анализаторы и системы машинного перевода, в корпусной лингвистики для изучения синтаксических явлений, в теоретической и психолингвистической практике — это доказательство взаимодействия.

ПропБанк Корпус (PropBank) более конкретно называемый «Банком предложений», представляет собой корпус, который снабжен устными предложениями и их аргументами[7,14].. Корпус — это ресурс, ориентированный на глагол; аннотации здесь более тесно связаны с синтаксическим уровнем. Мы можем использовать термин PropBank как обычное существительное, относящееся к любому корпусу, который аннотирован предложениями и их аргументами.

В области обработки естественного языка (NLP) проект PropBank сыграл очень важную роль. Это помогает в семантической ролевой маркировке.

VerbNet (VN),— это иерархический независимый от домена и самый большой лексический ресурс, на английском языке, который включает семантическую, и синтаксическую информацию о его содержимом[7,14]. VN —

это глагол широкого охвата, имеющий сопоставления с другими лексическими ресурсами, такими как WordNet, Xtag и FrameNet. Он организован в классы глаголов, расширяющие классы Левина путем уточнения и добавления подклассов для достижения синтаксической и семантической согласованности среди членов класса.

Каждый класс VerbNet (VN) содержит — наборы синтаксических описаний или синтаксических фреймов, семантических описаний, таких как анимация, человек, организация.

WordNet создан Princeton, представляет собой лексическую базу данных для английского языка. Это часть корпуса НЛТК. В WordNet существительные, глаголы, прилагательные и наречия группируются в наборы когнитивных синонимов, называемых **синсетами** . Все синтаксисы связаны с помощью концептуально-семантических и лексических отношений. Его структура делает его очень полезным для обработки естественного языка (НЛП).

В информационных системах WordNet используется для различных целей, таких как устранение неоднозначности слов, поиск информации, автоматическая классификация текста и машинный перевод. Одним из наиболее важных применений WordNet является выявление сходства между словами. Для этой задачи в различных пакетах были реализованы различные алгоритмы, такие как Similarity в Perl, NLTK в Python и ADW в Java.

Обработка естественного языка — Python

Для целей языковой обработки используем Python. Выбор инструментария определен нижеследующим:

Следующие функции отличают Python от других языков —

Python интерпретируется — нам не нужно компилировать нашу программу Python перед ее выполнением, потому что интерпретатор обрабатывает Python во время выполнения.

Интерактивный — мы можем напрямую взаимодействовать с переводчиком для написания наших программ на Python.

Объектно-ориентированный — Python является объектно-ориентированным по своей природе и облегчает написание этого языка на программах, поскольку с помощью этого метода программирования он инкапсулирует код в объектах.

Начинающий может легко учиться — Python также называют языком начинающего, потому что он очень прост для понимания и поддерживает разработку широкого спектра приложений.

Предпосылки

Последняя версия Python 3 выпущена на Python 3.7.1 и доступна для Windows, Mac OS и большинства версий ОС Linux.

Начало работы с NLTK

Будем использовать библиотеку Python NLTK (Natural Language Toolkit) для анализа текста на английском языке. Инструментарий естественного языка (NLTK, НЛТК) представляет собой набор библиотек Python, разработанный специально для идентификации и маркировки частей речи, встречающихся в тексте естественного языка, такого как английский.

Установка НЛТК

Установить NLTK, нам следует с помощью команды в среде Python

—pip install nltk

Если мы используем Anaconda, то пакет Conda для NLTK может быть собран с помощью команды

—conda install -c anaconda nltk

Загрузка данных НЛТК

После установки NLTK нужна загрузка предустановленных текстовых репозиториев, чтобы их можно было легко использовать. Предварительно нужно импортировать NLTK так же, как мы импортируем любой другой модуль Python. Следующая команда поможет нам в импорте NLTK

— import nltk

Далее загрузитб данные NLTK с помощью следующей команды

—nltk.download()

Установка всех доступных пакетов NLTK займет некоторое время.

Другие необходимые пакеты

Некоторые другие пакеты Python, такие как **gensim** и **pattern** , также очень необходимы для анализа текста, а также для создания приложений обработки естественного языка с использованием NLTK. пакеты могут быть установлены, как показано ниже —

gensim

gensim — это надежная библиотека семантического моделирования, которая может использоваться во многих приложениях. Мы можем установить его с помощью следующей команды —

pip install gensim

шаблон

Он может быть использован для **правильной** работы пакета **gensim** . Следующая команда помогает в установке шаблона —

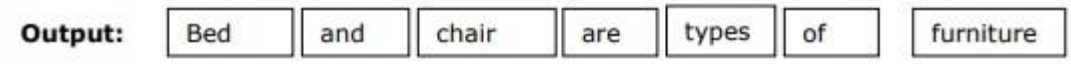
pip install pattern

лексемизацию

Токенизация может быть определена как Процесс разбиения данного текста на более мелкие единицы, называемые токенами. Слова, цифры или знаки препинания могут быть токенами. Это также можно назвать сегментацией слов.

пример

Вход — кровать и стул являются типами мебели.



У нас есть разные пакеты для токенизации, предоставляемые NLTK. Мы можем использовать эти пакеты в соответствии с нашими требованиями. Пакеты и детали их установки:

пакет sent_tokenize

Этот пакет можно использовать для разделения входного текста на предложения. Мы можем импортировать его с помощью следующей команды —

from nltk.tokenize import sent_tokenize

пакет word_tokenize

Этот пакет можно использовать для разделения входного текста на слова. Мы можем импортировать его с помощью следующей команды —

from nltk.tokenize import word_tokenize

Пакет WordPunctTokenizer

Этот пакет можно использовать для разделения входного текста на слова и знаки препинания. Мы можем импортировать его с помощью следующей команды —

from nltk.tokenize import WordPuncttokenizer

Морфологический

Из-за грамматических причин язык включает в себя множество вариаций. Вариации в том смысле, что язык, английский и другие языки тоже имеют разные формы слова. Например, такие слова, как ***демократия*** **,** ***демократия*** и ***демократизация*** **.** Для проектов машинного обучения очень важно, чтобы машины понимали, что эти разные слова, как и выше, имеют одинаковую базовую форму. Вот почему очень полезно извлекать базовые формы слов при анализе текста.

Стемминг — это эвристический процесс, который помогает в извлечении базовых форм слов путем измельчения их концов.

Ниже приведены различные пакеты для стемминга, предоставляемые модулем NLTK:

PorterStemmer пакет

Алгоритм Портера используется этим пакетом stemming для извлечения базовой формы слов. С помощью следующей команды мы можем импортировать этот пакет —

from nltk.stem.porter import PorterStemmer

Например, **«write»** будет выводом слова **«writing»,** заданным в качестве входных данных для этого стеммера.

LancasterStemmer пакет

Алгоритм Ланкастера используется этим пакетом stemming для извлечения базовой формы слов. С помощью следующей команды мы можем импортировать этот пакет —

from nltk.stem.lancaster import LancasterStemmer

Например, **«запись»** будет выводом слова **«запись»,** заданным в качестве входных данных для этого стеммера.

SnowballStemmer пакет

Алгоритм Snowball используется этим пакетом stemming для извлечения базовой формы слов. С помощью следующей команды мы можем импортировать этот пакет —

from nltk.stem.snowball import SnowballStemmer

Например, **«write»** будет выводом слова **«writing»,** заданным в качестве входных данных для этого стеммера.

лемматизации

Это еще один способ извлечь базовую форму слов, обычно целью которой является удаление флективных окончаний с помощью словарного и морфологического анализа. После лемматизации базовая форма любого слова называется леммой.

Модуль НЛТК предоставляет следующий пакет для лемматизации —

Пакет WordNetLemmatizer

Этот пакет извлечет основную форму слова в зависимости от того, используется ли оно как существительное или как глагол. Следующая команда может быть использована для импорта этого пакета —

from nltk.stem import WordNetLemmatizer

Подсчет POS-тегов – Chunking

Идентификация частей речи (POS) и коротких фраз может быть выполнена с помощью чанкинга. Это один из важных процессов в обработке естественного языка. Поскольку мы знаем о процессе токенизации для создания токенов, то на самом деле разбиение на фрагменты — это маркировка этих токенов. Другими словами, мы можем сказать, что мы можем получить структуру предложения с помощью процесса разбиения на фрагменты.

пример

В следующем примере мы реализуем чанки Noun-Phrase, категорию чанков, которые найдут чанки именных фраз в предложении, используя модуль Python NLTK.

Рассмотрим следующие шаги для реализации разбиения на именные фразы —

Шаг 1: Определение грамматики чанка

На этом этапе нам нужно определить грамматику для разбиения на фрагменты. Он будет состоять из правил, которым мы должны следовать.

Шаг 2: Создание парсера чанка

Далее нам нужно создать парсер чанка. Было бы разобрать грамматику и дать вывод.

Шаг 3: Выход

На этом шаге мы получим вывод в древовидном формате.

Запуск сценария НЛП

Начните с импорта пакета NLTK —

import nltk

Теперь нам нужно определить предложение.

Вот,

- DT является определяющим
- VBP это глагол
- JJ это прилагательное
- IN это предлог
- NN это существительное

DT является определяющим

VBP это глагол

JJ это прилагательное

IN это предлог

NN это существительное

```
sentence = [("a", "DT"),("clever","JJ"),("fox","NN"),("was","VBP"),
  ("jumping","VBP"),("over","IN"),("the","DT"),("wall","NN")]
```

Далее грамматика должна быть дана в форме регулярного выражения.

```
grammar = "NP:{<DT>?<JJ>*<NN>}"
```

Теперь нам нужно определить парсер для разбора грамматики.

```
parser_chunking = nltk.RegexpParser(grammar)
```

Теперь синтаксический анализатор проанализирует предложение следующим образом:

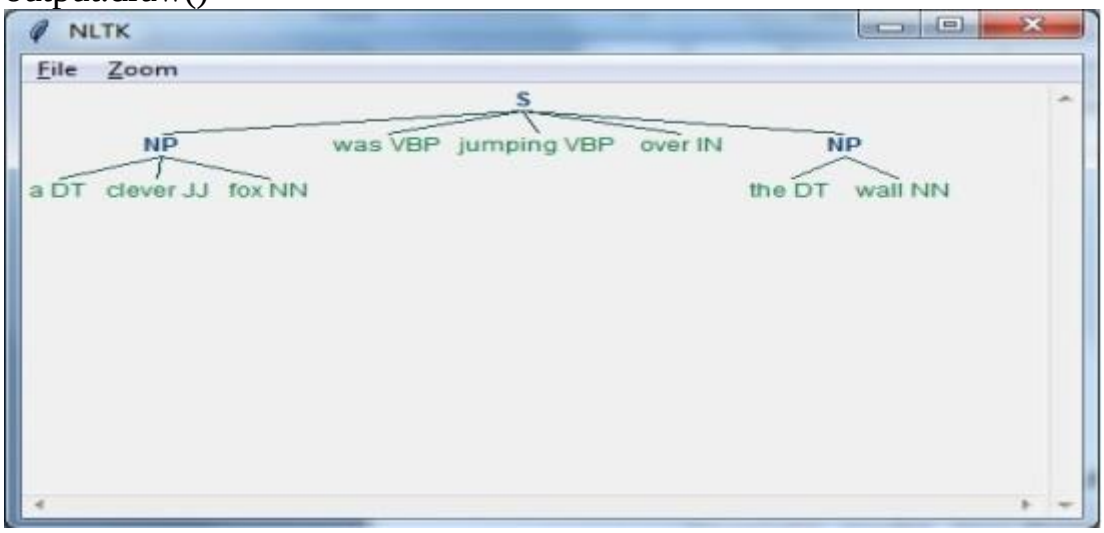
```
parser_chunking.parse(sentence)
```

Далее вывод будет в переменной следующим образом:

```
Output = parser_chunking.parse(sentence)
```

Теперь следующий код поможет вам сделать вывод в виде дерева.

```
output.draw()
```



2.5.4 Расстояние Левенштейна

Расстояние Левенштейна (редакционное расстояние, дистанция редактирования) — метрика, измеряющая по модулю разность между двумя последовательностями символов. Она определяется как минимальное количество односимвольных операций (а именно вставки, удаления, замены), необходимых для превращения одной последовательности символов в другую. В общем случае, операциям, используемым в этом преобразовании, можно назначить разные цены. Широко используется в теории информации и компьютерной лингвистике.

Впервые задачу поставил в 1965 году советский математик Владимир Левенштейн при изучении последовательностей [0,1], впоследствии более общую задачу для произвольного алфавита связали с его именем. Большой вклад в изучение вопроса внёс Дэн Гасфилд.

Расстояние Левенштейна дает возможность определить, как «похожие» две строки. Расстояние Левенштейна также называется «редактирование расстояния», которое точно описывает то, что он измеряет: количество измерений символов (вставка, удаление или замены), которые необходимы для

преобразования одной строки в другую. Интуиция заключается в следующем: чем меньше расстояние левенштейна, тем больше похожи на строки.

Расстояние Левенштейна и его обобщения активно применяется:

— для исправления ошибок в слове (в поисковых системах, базах данных, при вводе текста, при автоматическом распознавании отсканированного текста или речи).

— для сравнения текстовых файлов утилитой diff и ей подобными. Здесь роль «символов» играют строки, а роль «строк» — файлы.

— в биоинформатике для сравнения генов, хромосом и белков.

С точки зрения приложений определение расстояния между словами или текстовыми полями по Левенштейну обладает следующими **недостатками:**

При перестановке местами слов или частей слов получаются сравнительно большие расстояния;

Расстояния между совершенно разными короткими словами оказываются небольшими, в то время как расстояния между очень похожими длинными словами оказываются значительными.

Редакционным предписанием называется последовательность действий, необходимых для получения из первой строки второй кратчайшим образом. Обычно действия обозначаются так: D (англ. delete) — удалить, I (англ. insert) — вставить, R (replace) — заменить, M (match) — совпадение.

Например, для 2 строк «CONNECT» и «CONEHEAD» можно построить следующую таблицу преобразований:

M	M	M	R	I	M	R	R
C	O	N	N		E	C	T
C	O	N	E	H	E	A	D

Найти только расстояние Левенштейна — более простая задача, чем найти ещё и редакционное предписание (подробнее см. ниже).

Обобщения

Разные цены операций

Цены операций могут зависеть от вида операции (вставка, удаление, замена) и/или от участвующих в ней символов, отражая разную вероятность мутаций в биологии[3], разную вероятность разных ошибок при вводе текста и т. д. В общем случае:

W (a, b) — цена замены символа a на символ b

W (ε, b) — цена вставки символа b

W (a, ε) — цена удаления символа a

Необходимо найти последовательность замен, минимизирующую суммарную цену. Расстояние Левенштейна является частным случаем этой задачи при

w(a, a) = 0

w(a, b) = 1 при a≠b

w(ε, b) = 1

w(a, ε) = 1

Как частный случай, так и задачу для произвольных w, решает алгоритм **Вагнера — Фишера**, приведённый ниже. Здесь и ниже мы считаем, что все w неотрицательны, и действует неравенство треугольника: замена двух последовательных операций одной не увеличит общую цену (например, замена символа x на y, а потом y на z не лучше, чем сразу x на z).

Транспозиция

Если к списку разрешённых операций добавить транспозицию (два соседних символа меняются местами), получается расстояние Дамерау — Левенштейна. Для неё также существует алгоритм, требующий O(MN) операций. Дамерау показал, что 80 % ошибок при наборе текста человеком являются транспозициями. Кроме того, расстояние Дамерау — Левенштейна используется и в биоинформатике.

Формула

Здесь и далее считается, что элементы строк нумеруются с первого, как принято в математике, а не с нулевого, как принято во многих языках программирования.

Пусть S_1 и S_2— две строки (длинной M N соответственно) над некоторым алфавитом, тогда редакционное расстояние (расстояние Левенштейна) d (S_1,S_2)можно подсчитать по следующей рекуррентной формуле

$\mathrm{d}(S_1, S_2) = D(M, N)$, где

$$D(i,j) = \begin{cases} 0, & i = 0,\ j = 0 \\ i, & j = 0,\ i > 0 \\ j, & i = 0,\ j > 0 \\ \min\{ & \\ \quad D(i, j-1) + 1, & \\ \quad D(i-1, j) + 1, & j > 0,\ i > 0 \\ \quad D(i-1, j-1) + \mathrm{m}(S_1[i], S_2[j]) & \\ \} & \end{cases}$$

где m(a,b)=0 , если a=b

и m(a,b)=1 в противном случае;

min {a,b,c}возвращает наименьший из аргументов.

Здесь шаг по i символизирует удаление (D) из первой строки, по j — вставку (I) в первую строку, а шаг по обоим индексам символизирует замену символа (R) или отсутствие изменений (M).

Очевидно, справедливы следующие утверждения:

- $\mathrm{d}(S_1, S_2) \geqslant \big||S_1| - |S_2|\big|$
- $\mathrm{d}(S_1, S_2) \leqslant \max\big(|S_1|, |S_2|\big)$
- $\mathrm{d}(S_1, S_2) = 0 \Leftrightarrow S_1 = S_2$

Расстояние Левенштейна имеет важные приложения. Подумайте о функциональности автоматической коррекции на вашем смартфоне. Скажем, вы вводите «Helo» в своем WhatsApp Messenger. Затем ваш смартфон выбирает несколько высоких слов вероятности и сортирует их (например, расстояние левенштейна). Например, тот, с минимальным расстоянием левенштейна (и, следовательно, максимальное сходство) – это строка «Hello». Таким образом, он может автоматически исправить «Helo» на «Hello».

Давайте рассмотрим пример с двумя строками «кошка» и «Черо». Как рассчитать расстояние Левенштейна в этом сценарии?

Мы уже знаем, что расстояние Levenshtein вычисляет минимальное количество редактиций (вставка, удаление или замену) для достижения второй строки, начиная с первой строки.

Вот одна минимальная последовательность:

“Кот”

« ЧТ » (заменить «А» по «Н»)

« че » (заменить «т» по «е»)

” Чел ” (Вставка “L” в положении 3)

” Chell ” (Вставить “L” в положении 4)

« Chello » (Вставьте «О» в положение 5)

Каким образом, мы можем преобразовать строку «кошку» в строке «Chello» в пять шагов редактирования – расстояние левенштейна составляет 5.

Код на Python для расчета расстояния Левенштейна текста из двух строк A и B приведен на рис 9.1.

```
## The Data
a = "cat"
b = "chello"
c = "chess"

## The One-Liner
ls = lambda a, b: len(b) if not a else len(a) if n
        else min(ls(a[1:], b[1:])+(a[0] != b[0]),
                 ls(a[1:], b)+1,
                 ls(a, b[1:])+1)

## The Result
print(ls(a,b))
print(ls(a,c))
print(ls(b,c))
```

Рис. 9.1 - Код на Python для расчета расстояния Левенштейна текста из двух строк A и B

Исследуем важный методы Python, который здесь исспользован. В Python каждый объект имеет значение правды –либо правда, либо ложь в мире Python. Большинство объектов на самом деле верны. Интуитивно, вы знаете, что несколько объектов, которые являются ложными, не так ли?

0 ложно

“Ложно

[] ложно

{} ложный

Как правило, объекты Python считаются ложными, если они пусты или нуля. Оснащен этой информацией, теперь вы можете легко понять первую часть функции левенштейна:

Мы создаем функцию лямбда, которая возвращает количество изменений, необходимых для преобразования строки в строка б. Существует два тривиальных случая: Предположим, что строка A пустая. В этом случае минимальное расстояние редактирования – Лен (б) Вставки символов в строке b. Мы не можем сделать лучше. Точно так же, если строка B пуста, минимальное расстояние редактирования – Лен (а). Таким образом, мы можем напрямую вернуть правильное расстояние редактирования, если ни один из строк пуст.

Покажем, обе строки не пустые (в противном случае решение тривиально, как показано ранее). Теперь мы можем упростить проблему тремя способами.

Во-первых, мы игнорируем ведущие символы обоих строк A и B и рассчитайте расстояние редактирования от [1:] до B [1:] в рекурсивном порядке. Если ведущие символы A [0] и B [0] разные, мы должны исправить его, заменив

[0] на B [0]. Следовательно, мы увеличиваем расстояние редактирования на один, если они отличаются.

Во-вторых, мы удаляем первый персонаж A [0]. Теперь мы проверяем минимальное расстояние редактирования рекурсивно для этой меньшей проблемы. Как мы убрали персонаж, мы увеличиваем результат на один.

В-третьих, мы (концептуально) вставьте символ b [0] к началу слова a. Теперь мы можем уменьшить эту проблему в меньшую проблему, которая возникает, если убрать первый символ б. Как мы выполнили одну операцию редактирования (вставку), мы увеличиваем результат на одну.

Наконец, мы просто принимаем минимальное расстояние редактирования всех трех результатов (замените первый символ, удалите первый символ, вставьте первый символ).

Это решение еще раз демонстрирует важность подготовки ваших навыков рекурсии – рекурсия может не прийти к вам естественным путем, но отдыхать уверены, что он будет изучать многие рекурсивные проблемы, такие как этот.

Это продвинутый алгоритм, который требует основных компьютерных наук и навыков Python. Если вы чувствуете, что вам нужно тренировать основы, прочитайте книгу «Кофе-брейк Python». После изучения книги вы не только знаете свой точный уровень квалификации Python, вы также поймете код Python намного быстрее!

Работая в качестве исследователя в распределенных системах, директор Кристиан Майер нашел свою любовь к учению студентов компьютерных наук.

Чтобы помочь студентам достичь более высоких уровней успеха Python, он основал сайт программирования образования Finxter.com Отказ Он автор популярной книги программирования Python One-listers (Nostarch 2020), Coauthor of Кофе-брейк Python Серия самооставленных книг, энтузиаста компьютерных наук, Фрилансера и владелец одного из лучших 10 крупнейших Питон блоги по всему миру.

2.5.5 Вопросы к лекции№9

1. Опишите схему семантической нейронной сети, выполняющей морфологический и синтаксический разбор как синхронизированное линейное дерево
2. Опишите принцип действия семантической нейронной сети, выполняющей морфологический и синтаксический разбор как синхронизированное линейное дерево
3. Какие НЛП — лингвистические ресурсы вам известны
4. Расскажите о методах Python применимых для обработки текстов на естественном языке.
5. Расскажите о алгоритм подсчета расстояния Левенштейна
6. Продемонстрируйте компьютерную реализацию алгоритма подсчета расстояния Левенштейна.
7. Как рассчитать минимальное расстояние редактирования.

Библиографический список

1. Бенгфорт Бенджамин, Билбро Ребекка, Охеда Тони Б46 Прикладной анализ текстовых данных на Python. Машинное обучение и создание приложений обработки естественного языка. — СПб.: Питер, 2019. — 368 с.: ил. — (Серия «Бестселлеры O’Reilly»)
2. Бодянский Е.В., Руденко О.Г. Искусственные нейронные сети: архитектуры, обучение, применения. Харьков: ТЕЛЕТЕХ. 2019. 369 с.
3. Браславский П. «Введение в обработку естественного языка». [видеокурс]. URL: https://stepik.org/course/1233/promo (дата обращения: 31.01.2020).
4. Бринк Х., Ричардс Д., Феверолф М. Машинное обучение. СпБ.: Питер. 2017.
5. Вьюгин В.В. Математические основы машинного обучения и прогнозирования. М.: МЦНПО. 2017.
6. Дубаков А.А. Введение в объектно-ориентированное программирование на Java: учеб. пособие. – СПб: НИУ ИТМО, 2016. – 250 с. Электронно-библиотечная система «Лань»: [сайт]. URL: https://e.lanbook.com/book/110468 (дата обращения: 31.01.2020).
7. Ингерсолл, Г. Грант, С. Ингерсолл Обработка неструктурированных текстов. Поиск, организация и манипулирование / Грант С. Ингерсолл, Томас С. Мортон, Эндрю Л. Фэррис ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2015. - 414 с. - ISBN 978-5-97060-144-0. - Текст : электронный. - URL: https://znanium.com/catalog/product/1027786 (дата обращения: 24.06.2021).
8. Лукьяненко Т. В. Основы теории управления : учеб. пособие / Т. В. Лукьяненко, Н. П. Орлянская. – Краснодар : КубГАУ, 2018. – 94 с.
9. Котельников Е. В. Распознавание эмоциональной составляющей в текстах: проблемы и подходы / Е. В. Котельников, М. В. Клековкина, Т. А. Пескишева, О. А. Пестов; под. ред. С. М. Окулова. - Киров: Изд-во ВятГГУ, 2019. - 103 с.
10. Круглов В.В., Борисов В.В. Искусственные нейронные сети. Теория и практика. М.: Горячая линия – Телеком. 2019. 382 с.
11. Маккинли, У. Python и анализ данных / Уэс Маккинли ; пер. с англ. А.А. Слинкина. - Москва : ДМК Пресс, 2016. - 482 с. - ISBN 978-5-97060-315-4. - Режим доступа: http://znanium.com/catalog/product/1027796
12. Маннинг К. Д., Рагхаван П., Х. Шютце Введение в информационный поиск – Вильямс, 2018. 246 с.
13. Метрики качества ранжирования /Блог компании E-Contenta / Хабр https://habr.com
14. Обработка естественного языка https://intellect.icu/analiz-i-indeksirovanie-dannykh-i-obrabotka-zaprosa-invertirovannyj-indeks-6236
15. Орлянская Н. П. Информационные технологии : учеб. пособие / Н. П. Орлянская. – Краснодар : КубГАУ, 2020. – 139 с.
16. Принцип работы поисковых систем Яндекс и Google: релевантность и факторы ранжирования сайта https://labrika.ru/blog/informacionnii-poisk

17. Шарден Б., Массарон Л., Боскетти А. «Крупномасштабное машинное обучение вместе с Python» издательство «ДМК Пресс», 2018 – 358 стр
18. Янаева М.В. Нечеткие нейросети в интеллектуальном анализе данных / М.В. Янаева, Е.В. Синченко // Электронный сетевой политематический журнал «Научные труды КубГТУ»
19. Liu B. Sentiment analysis and Subjectivity // Handbook of Natural Language Processing. CRC Press, Taylor and Francis Group, Boca Raton, 2017. P. 1-38.
20. Making Sentimental Analysis Easy With Scikit-Learn [Электронный ресурс]. URL: 0
21. Mor, M., and Fraenkel, A.S., A hash code method for detecting and correcting spelling errors, Communications of the ACM, 1982 (http://portal.acm.org/citation.cfm?id=358752), а также Pollock, J.J., and Zamora, Automatic spelling correction in scientific and scholarly text, Journal of the American Society for Information Science and Technology (JASIST), 1984, (http://portal.acm.org/citation. cfm?id=358048)
22. Dragomir R. Radev, John Prager, and Valerie Samn. Ranking suspected answers to natural language questions using predictive annotation. In Proceedings of the 6th Conference on Applied Natural Language Processing, Seattle, WA, May 2000.
23. Hovy, E., Gerber, L., Hermjakob, U., Junk, M. & Lin, C. (2000) Question Answering in Webclopedia. In: 9th Text Retrieval Conference.
24. Huettner, A. (2000) Question Answering. In: 5th Search Engine Meeting.John Prager, Eric Brown, Anni Coden, and Dragomir Radev. Question-answering by predictive annotation. In Proceedings, 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, Athens, Greece, July 2000.
25. Katz, B., Felshin, S. & Lin, J. (2002) The START Multimedia Information System: Current Technology and Future Directions. In: International Workshop on Multimedia Information Systems.
26. Wong, W. (2005) Practical Approach to Knowledge-based Question Answering with Natural Language Understanding and Advanced Reasoning. In: Master; National Technical University College of Malaysia.