МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
(СГУ)

ВВЕДЕНИЕ В ИНФОРМАЦИОННЫЙ ПОИСК

Студентов 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ

Проверил
доцент, к. ф.-м. н. _____ А. А. Кузнецов

Саратов 2018

СОДЕРЖАНИЕ

ВВЕДЕНИЕ 3
1 Булев поиск, обратный индекс, разбор текста 4
1.1 История 4
1.2 Базовые понятия (учить не надо, но полезно) 5
1.3 Булев поиск 6
1.4 Матрица инцидентности 6
1.5 Обратный индекс 7
1.6 Разбор документа 8
1.7 Быстрое пересечение списков 10
1.8 Исправление ошибок 11
2 Сбор данных с web 12
3 Обзор Lucene 15
4 Построение поискового индекса 18
5 Лекция 5 20
5.1 Поиск по * и перестановочный индекс 20
5.2 Поиск по числовым полям 20
5.3 Геопоиск 20
5.4 Сжатие индекса 20
5.5 Сжатие инвертированного индекса 21
6 Ранжирование 22
6.1 Зональное ранжирование 22
6.2 Частота термина и взвешивание 22
7 Оценка качества поиска 24
8 Вероятностная модель 25
9 Языковые модели 26
10 Современные языковые модели 27
11 Синонимия 29
12 Машинное обучение в ранжировании 30
13 Learning to rank 33
13.1 Попарный подход 33

ВВЕДЕНИЕ

Задание 1. Исправление опечаток. Есть словарь с правильными словами и есть документ с опечатками. Взять готовый словарь на 600 тысяч – миллион слов. Попробовать и сравнить два варианта алгоритма – простой и с k-грамм индексом.

Задание 2. Выбрать сайт, в котором есть какой-нибудь каталог (книжек, фильмов) (5-10 тысяч элементов). Написать краулер, который соберет информацию с этого сайта, которую будем использовать для построения поискового индекса. Желательно, чтобы о товара было несколько полей.

Задание 3. Создать индекс с помощью Lucene – записать все документы в индекс и что-нибудь поискать.

Задание 4. Посмотреть, каким образом можно добавить и удалить документ из люценовского индекса и сразу попробовать найти этот документ, если сразу не будет искаться – надо будет разобраться, в какой момент документ станет доступным (либо сделать коммит, либо еще что-то).

Задание 5. Добавить поиск по числовым полям.

Задание 6. Посмотреть, как автомат работает в Lucene. Нужно засунуть наш словарь в этот автомат.

Задание 7. Сделать assesment (для документ + запрос сделать оценки, от 0 до 2) и оценить качество поисковой системы. Использовать $NDCG$.

Задание 8. Построить вероятностную модель по нашему индексу. Одограммную (вообще без условной вероятности), двуграммную и трехграммную модель. Попробовать для каждой из этих моделей продолжить фразу.

Задание 9. Обучить нейросеть Word2Vec, Glove или эмебдинг на нашем корпусе товаров и посмотреть похожие ли слова.

Задание 10. Определить, какие есть синонимы в нашем домене. Добавить поиск по синонимам. Классификатор можно не делать, реализовать tf-idf, pmi, близость в векторном пространстве.

Задание 11. Улучшить поиск с помощью машинного обучения.

1 Булев поиск, обратный индекс, разбор текста

1.1 История

Способы передачи знания:

1. Устные предания.
2. Письменность.
3. Библиотеки.
4. Интернет.

История письменности:

1. Глиняные таблички (5000-3000 до н.э.) - в основном бухгалтерский учет.
2. Алфавит (2000-1500 до н.э.).
3. Печатный станок (в 15 веке).

Традиционный поиск:

— Объем 1mb-10gb.
— Поиск перебором.
— Время отклика – несколько секунд.
— Один пользователь.

Современный поиск:

— Объем – весь интернет.
— Сложные алгоритмы.
— Время отклика – 10-500 мс.
— Миллионы пользователей.

Развитие поисковых систем

— 29 окт 1969г - рождение интернета (передача данных между двумя компьютерами на разных концах Америки)
— 6 авг 1991г - первый сайт info.cern.ch. . . .., т.к. некоторые данные нужны были сразу группе лиц
— июнь 1993г - страница “What’s new” о новых сайтах и краткое описание, поддерживалась вручную
— 1993г - первая поисковая система AliWeb (владельцы сайтов на спец страницах хранили список документов с указанием о том, какие именно данные можно найти в тех или иных документах; поисковая система использовала эти данные, каталогизировала и использовала для запросов пользователей; 2 минуса: нужен список сайтов, на которых искать спец страницы + сами данные вручную определялись владельцами)

— 1994г - keyword-based поиск (“найди мне то, что я сказал”)
— 1998г - ранжирование PageRank (“найди мне то, что я имел в виду”) - учитывает авторитетность документа

1.2 Базовые понятия (учить не надо, но полезно)

Информационный поиск - процесс поиска ответа среди неструктурированной документальной информации, призванного удовлетворить информационную потребность пользователя

Информация - сведения независимо от формы их представления (в нашем случае: документы - носители информации)

Запрос - сформулированная пользователем поисковая потребность

Результаты поиска - ответ поисковой системы на запрос пользователя

Документ - объект, содержащий в себе информацию (из него извлекаются данные, по кот осущ поиск, он может быть ответом на поисковый запрос пользователя)

Корпус - набор документов, кот исп для построения поисковой системы

Релевантность - способность конкретного документа успешно удовлетворить инф потребность пользователя, выраж в форме конкретного запроса. Этот термин используется только в контексте пары документ-запрос.

Полнотекстовый поиск - это поиск, который в ходе построения ответа пользователю основывается в первую очередь на том, какой текст хранится в самом документе.

Основные характеристика качества поиска:

— Точность - сколько документов из числа найденных поиск системой действительно являются релевантными запросу пользователя
— Полнота - сколько документов из всех, что релевантны запросу пользователя, смогла найти поиск система

Классификация поисковых систем (учить не надо)

— по хранению данных для поиска (автономная / внешняя (веб-поиск))
— по типу данных для поиска (текст / нетекстовые данные (видео, аудио, картинка) / метаданные (рег номера, дата создания))
— по типу корпуса (корпус тематич документов / корпус “случ” документов)
— по виду запросов (текстовый ввод / голосовой ввод / по образцу (картинке) / альтерн)

— по результату (набор документов / все релевантные документы / информация (место на карте) / результат вычислений (построенный маршрут)))

1.3 Булев поиск

Булев поиск – есть 3 оператора (булева алгебра):

— AND - A AND B - документ содержит и A, и B
— OR
— NOT – используется в чистом виде редко, в основном AND NOT.

Сложный запрос: “лук” AND (“стрельба” OR “стрельбище”) AND NOT “растение” Наивный подход:

— найти все док где “лук”
— найти все док где стрельба
— где стрельбище
— операция OR
— операция AND и т.д. . . (поэтапно)

Очевидно, что каждый раз использовать весь корпус документов для обработки одного запроса пользователя долго и накладно.

1.4 Матрица инцидентности

Матрица термин-документ: столбцы – названия документов (книг), строки – термины, единица на пересечении, если это слово присутствует в документе (рис. 1).

	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 – Матрица инцидентности

Если хотим найти документ, в котором встречаются несколько слов –

берем AND всех этих строк, если те, в которых есть хоть какое-то из слов – OR и т.д (рис. 2).

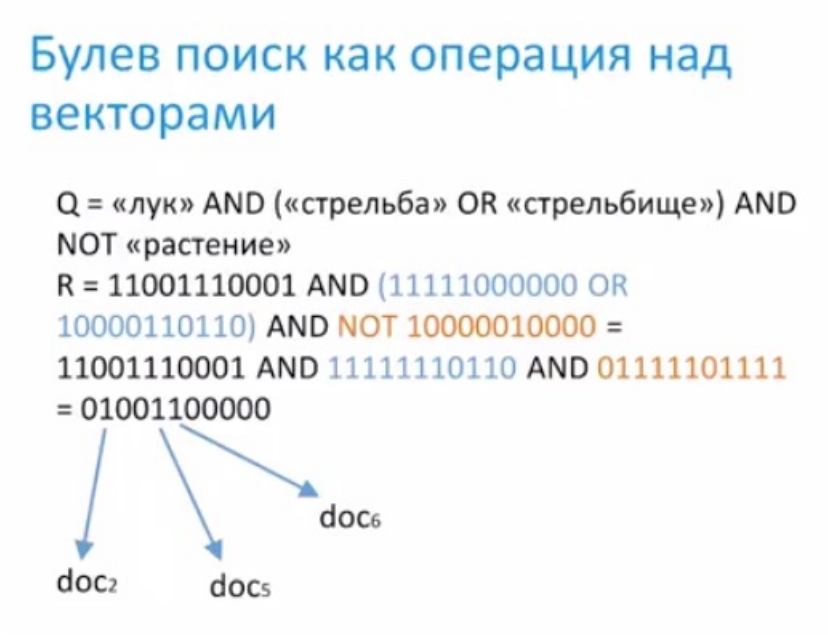

Рисунок 2 – Булев поиск как операция над векторами

Матрица инцидентности в общем случае сильно разрежена => можем хранить только значимые части, т.е. обратный индекс

1.5 Обратный индекс

Обратный индекс (рис. 3):

- Для каждого термина собираем документы, где это слово встречается, заменяем документы на их ID.
- Получаем для каждого термина отсортированный список документов, где это слово встречается. Размер таких списков будет расти линейно и занимать не больше места, чем исходные документы.

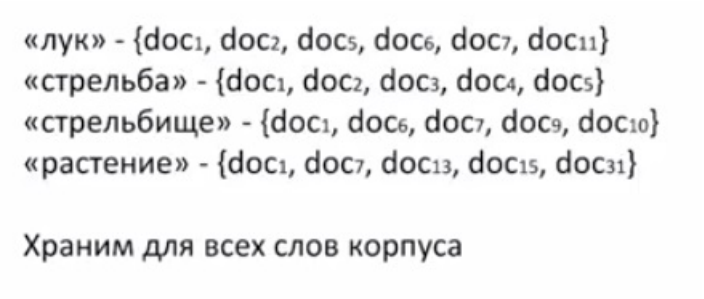

Рисунок 3 – Обратный индекс

Что ещё можем хранить:

— позиция слова в документе
— частота слова в корпусе
— популярность документа

1.6 Разбор документа

Чтобы построить индекс, надо сначала его обработать 4.

— Выделение терминов.
— Нормализация.

Рисунок 4 – Разбор документа

Токен - экземпляр последовательности символов в документе, объединенных в семантическую единицу для обработки

Терм - “нормализованный” токен (регистр, морфология, исправление ошибок)

Нормализация зависит от языка документа:

— mit - слово на немецком,
— MIT - университет. Такие случаи надо обрабатывать по-разному

Регистр

— toLower(all)
— Возможны исключения (MIT и mit)
— часто лучше понижать все, т.к. пользователя не заботятся о капитализации в запросах

Точки, запятые, тире и другие знаки. В некоторых случаях исключать из индексов, но возникают проблемы.

Стоп-слова есть почти во всех документах коллекции, например, the, be, and, of, a. Их исключают из индексов.

Что делать, если эти слова несут смысловую нагрузку? Используются *биграммы* – слова разбиваются не по одному, а по парам, тогда стоп-слово не встречается во всех словах. Но используется это довольно редко.

Stanford University - булев поиск по одному слову может дать не те результаты, поэтому нужно использовать биграммы Нужно помнить, что при изменении длины граммы, индекс увеличивается

Positional index: для каждого термина из лексикона хранятся словопозиции: слово: docID: сколько раз: <pos1, pos2, ...>. Для поиска фразы можно учитывать расстояние (=1). Исп такой индекс, можно расширить возможности запросов, указывая макс расстояние между словами существенно увеличивает объем

Для лучшего результата использовать смешанный подход: Britney Spears - встречается редко, исп positional index; The Who <- встречается часто, исп биграмму.

Проблемы токенизации в примерах:

- Hewlett-Packard - имена собственные
- State-of-the-art - устойчивое выражение
- Co-education - слово с тире
- San Francisco - нельзя разбивать
- York University vs. New York University
- обработка чисел, дат, телефонных номеров, ip address - разные форматы
- в китайском нет пробелов, в немецком компаунды и т.д.
- .NET, O'Reilly - учитывать знаки
- стоп-слова - слова, кот есть почти во всех документах (the, be, and, of, a)
- Named Entity Recognition - извлечение объектов, кот означают одно понятие: ФИО, адреса, телефоны, даты, названия песен, фильмов и т.д.

Стемминг (нормализация) – например, creates, created, creating преобразовать в creat. Стемминг просто находит у слова окончания и выкидывает его. Для поиска окончаний есть ряд правил, работает хорошо для английского языка. В русском работает очень плохо, поэтому используются другие алгоритмы (Стеммер Портера, Stemka, Mystem).

Ошибки стемминга: run, men, children – на этих словах стемминг не работает, в таких случаях лучше использовать морфологию.

Лемматизация - привести все разные формы одного слова к начальной (каноничной). Заключается в поиске начальной формы (леммы в словаре).Обычно используется конечный автомат. Что делать со словами, которых нет в словаре? Ищем похожие!

Слово = машинная основа + парадигма (окончание)

Проблемы: омонимия: белки (белка /белок); словоформа (run, men, children - конкретная морфологическая разновидность слова)

Морфология:

— Строится по словарю.
— Для незнакомых слов используется эвристика, позволяющая находить нормализированную форму слова.

Процесс нормализации, реализованный в грамматическом словаре, позволяет убрать из исходного текста грамматическую информацию (падежи, числа, глагольные виды и времена, залоги причастий, род и так далее), оставляя смысловую составляющую.

Два других алгоритма - стемминг и лемматизация - пытаются достичь такого же эффекта, но глубина преобразования текста в них меньше. Другая сторона медали - более существенные затраты вычислительных ресурсов на выполнение всех стадий алгоритма глубокой нормализации.

В качестве примера можно взять предложение *вижу три села*. При неблагоприятном стечении обстоятельств нормализация даст на выходе малорелевантный результат: *видеть тереть сесть* вместо корректного *видеть три село*.

1.7 Быстрое пересечение списков

Быстрое пересечение списков 5:

— Сложность n+m.
— Если пересечение маленькое, то можно сделать быстрее.
— Можно идти не просто двумя указателями, но при несовпадении каждый раз увеличивать шаг в 2 раза.
— Также можно разбивать на $\sqrt{n}$ частей и искать в них.

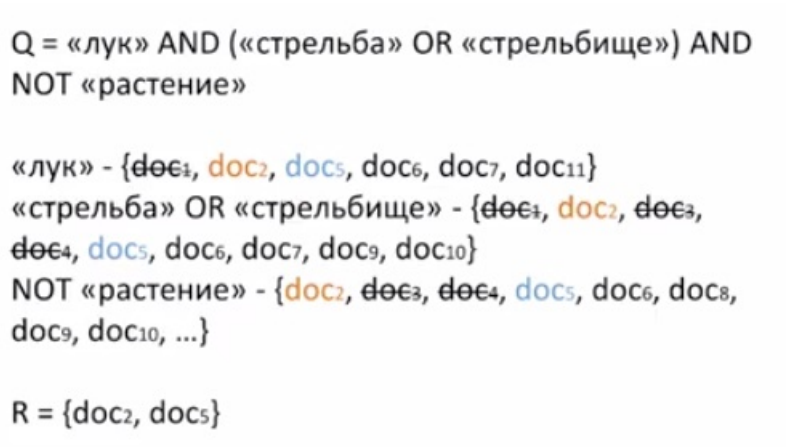

Рисунок 5 – Обратный индекс

1.8 Исправление ошибок

Кандидаты на исправление:

— словарь известных слов
— слово из словаря
— ближайшее по какой-нибудь метрике

В качестве метрики будем использовать расстояние Левенштейна – минимальное количество символов, которые нужно поменять, добавить, удалить, чтобы из одного слова получить другое, решается с помощью ДП со сложность $O(n \cdot m)$

Первый вариант: перебрать все слова в словаре, для каждой пары подсчитать расстояние. Работает долго. Нужно сузить количество кандидатов.

Второй вариант: k-грамм индекс – каждое слово разбиваем на k-граммы (по k символов, например опечатка на опе, печ, еча, чат, тка). Для каждой k-граммы список слов, где она встречается. Например, опе -> . . . , операция, опечатка, . . .

Нахождение кандидатов – объединить списки k-грамм, можно брать только те слова, у которых большой процент совпадения k-грамм. Выбираем топ-100 слов с наибольшим весом.

2 Сбор данных с web

Устройство web:

— HTTP протокол.

— Ссылки.

На основе ссылок строится web-граф.

Структура URL: <протокол>://<логин>:<пароль>@<хост>:<порт>/<URL-путь>?<параметры>#<якорь>

Протоколы:

— http(s)

— ftp

— mailto

— irc

<логин>:<пароль>

— user

— user:password

<хост>:<порт>

— localhost:8080

— sgu.ru

— 192.168.1.1

<URL-путь>

— somedir/somefile.html

case-(in)sensetive – маленькие и большие в названиях файлах равнозначны – нельзя создать два файла, один с большой буквы, а другой с таким-же названием, но с маленькой.

<параметры>#<якорь>

— someanchor – в URL не передается

Протокол http – сервер последовательно получает:

— Стартовая строка:
  - GET index HTTP HTTTP/1.0
  - Host: sgu.ru

— Заголовки:
  - Responce headers
  - Request headers

Content type:

- html/text
- application/json
- application/octet-stream
- image/gif

HTTP status code:

- 200 OK
- 201 Created
- 202 Accepted
- 204 No content
- 301 Moved Permanently (Location header)
- 302 Found, 302 Moved Temporarily
- 304 Not modified – с последнего обращения ничего не изменилось
- 400 Bad Request
- 401 Unauthorized
- 403 Forbidden – уже авторизовались, но прав не хватает
- 404 Not Found
- 500 Internal Server Error
- 501 Not Implementrd – функциональность еще не реализована
- 503 Service Unavailable
- 504 Gateway Timeout

— Тело сообщения:

HTML – логическое описание документа.

CSS – описание внешнего вида документа.

HTML A tag – тэг со ссылкой.

Селектор – описание, каким образом мы выберем элементы.

Виды селекторов (первые 4 одинарные, дальше – каскадные) (код в презентации):

- div –
- .note – выбор по классу
- #par – выбор по уникальному id
- a[href="test"]
- div#par – выбрать div, а в нем найти по id
- p.note – выбрать все элементы p с классом note
- p.note>b – внутри еще выбрать тег b

Разбор страницы (библиотеки):

- jQuery (JS)
- JSOUP
-

Инструментарий.

Поисковый робот:

- Устойчивость – ошибки сайта, неправильные статусы не ломают работу робота.
- Вежливость – робот не делает много запросов, чтобы не сломать сайт (надо ориентироваться на популярность сайта).

Схема работы поискового робота: Очередь URL $\rightarrow$ Скачать страницу $\rightarrow$ Анализ $\rightarrow$ Разобрать содержимое $\rightarrow$ Извлечь URL $\rightarrow$ Проверить уникальность URL (или если прошло много времени и надо скачать снова) $\rightarrow$ Очередь URL.

3 Обзор Lucene

Первый релиз в марте 2000.

Основные концепции:

- Document – то, что индексируется и будет возвращаться в поиске.
- Field – поле, у документа может быть много полей с одинаковыми названиями.
- Term – разбиваем поле на термы.

Field Type:

- Indexed – по этому полю будет осуществлен поиск.
- Stored – поле будет хранить в базе, его можно будет достать.
- Tokenized – поле будет разбиваться на токены и храниться в базе.

Типы полей:

- Text – токенизируется.
- String – не токенизируется.
- Double – поддерживается поиск по числовым полям.
- Float
- Long
- Integer

Код создания документа в презентации.

commit – запись данных на диск (не в ОП). В некоторых случаях до такой записи нельзя искать в документе.

Long merge tree – используется для организации данных.

Стадии добавления документа:

- Tokenizer – разбивает текст на токены.
- Token Filter (возможно несколько) – выполняет преобразования над токенами (приводит все к нижнему регистру, разбивает на k-граммы, приводит к стандартному виду и т.д.)
- Index

Tokenizerы:

- StandardTokenizer – умно все разбивает, находит точки, тире, определяет концы предложения, но при этом сохраняет фразы вроде .NET
- KeywordTokenizer – ничего не разбивает, но ищет ключевые слова
- PatternTokenizer – настраивается по своим паттернам.

TokenFilters:

— LowerCaseFilter – приводит к нижнему регистру.
— StopFilter – позволяет останавливать поиск после нахождения стоп-слова.
— ..

Запросы:

— Query – класс, от которого унаследованы все остальные запросы.
— TermQuery – поиск по терму, найти все документы, у которых есть данный терм. При создании нужно указать имя поля и значение этого поля.
— BooleanQuery – позволяет соединять несколько TermQuery с помощью булевого поиска, конструкции:
  – SHOULD – должен присутствовать поиск, но не обязателен.
  – MUST
  – MUST_NOT
— PhraseQuery – позволяет учитывать количество слов между найденными словами.
— TermRangeQuery – найти все слова, начинающие с определенной последовательности букв (от ab до ac например).
— PrefixQuery
— WildcardQuery – поиск со звездочкой в слове
— RegExpQuery – поиск по регулярному выражению, но нельзя, чтобы звездочка стояла в начале слова.

Есть queryParser, позволяющий распарсить строку на языке запросов. Но лучше писать свой трансформатор под определенную структуру.

Язык запросов:

— field:term – поле и терм
— field:«phrase query» – если хотим искать по фразе – нужны кавычки
— «phrase query» – искать везде фразу, если указать без кавычек – превратится в or query?
— Term modifers: te?t, test*, te*t, но нельзя *est, ?est
— Fuzzy Search esroam 1 – все термы, для которых расстояние Левенштейна <= 1.
— Range Searches mod_date:[20020101 TO 20030101] – поиск в диапазоне.
— NOT OR AND
— + – must, - – not, по умолчанию should

Задание 3. Создать индекс с помощью Lucene – записать все документы

в индекс и что-нибудь поискать.

4 Построение поискового индекса

Построение индекса:

— Построение в памяти – как в задании 1. Проблемы: хранится в ОП, ее может не хватать.
— Блочное индексирование – если коллекция не помещается в память, разбиваем на блоки, сохраняя их на диске. При объединении сначала сортируем блоки, а затем их сливаем.
— Распределенное индексирование – индекс не помещается на диске одной машины. Кроме того процесс построение индекса может занимать много времени. Исходные документы разбиваются на пачки, каждая машина заведует своей пачкой. Можно индексировать, например, на 50 машинах, а поддерживать индекс всего на 3 самых мощных.
— Map reduce – строит единый индекс, который должен помещаться на одну машину. Используется редко. Это парадигма параллельных вычислений:
  – split – разделяет документы по пачкам, каждому документу присваивается id.
  – map – разделяем документы на токены, применяем токенайзер, аналайзер.
  – shuffle – проводим сортировку информацию по словам, затем по индексу.
  – reduce – объединяем списки, получаем готовый обратный индекс.
— Динамическое индексирование – в памяти индекс строится динамически, когда он достигает определенного объема – он записывается на диск.

  Для обеспечения надежности одновременно идет запись в индекс памяти и во write ahead log на диске.

  Чтобы не бегать по большому количеству блоков, их нужно время от времени сливать. Это можно делать в фоновом процессе, одним из ядер, пока остальные занимаются построением индекса.

  Некоторые документы могут оказаться сразу в нескольких блоках, но в некоторых из них они будут помечены как удаленные. Действительно удаление произойдет только при слияние блоков.

  Организация поиска:

— Поисковый кеш:
  – Кеш документов.

  - Кеш запросов.
  - Кеш полей – сохраняем часто употребляемые слова.
- Индекс в памяти.
- Блоки на диске.
- WAL (write ahead log).

Коммит – заливает индекс из памяти на диск, а оптимайз сливает блоки в памяти.

Задание 4. Посмотреть, каким образом можно добавить и удалить документ из люценовского индекса и сразу попробовать найти этот документ, если сразу не будет искаться – надо будет разобраться, в какой момент документ станет доступным (либо сделать коммит, либо еще что-то).

5 Лекция 5

5.1 Поиск по * и перестановочный индекс

Дописываем в конце слова какой-нибудь специальный символ, например знак доллара. Кроме того размножаем каждое слово.

test:

- test$
- est$t
- st$te
- t$tes
- $test

Для поиска n*m преобразуем: $\rightarrow$ n*m$ $\rightarrow$ m$n*

5.2 Поиск по числовым полям

Нужна поддержка поиска по числам в диапазонах.

Для индексации числового значения вместо числа сложим в индекс набор чисел, всякий раз увеличивая количество знаков после запятой. Например для π: 3, 3.1, 3.14, 3.141 и т.д.

Затем для поиска в интервале выбираем подходящее количество знаков после запятой и берем все такие числа в интервале. Например для поиска от 2 до 3.5 возьмем поиск по 2 и 3, для от 3.1 до 3.25 – 3.1 и 3.2, для от 3.1 до 3.15 – 3.1. После этого лишние отфильтровываем.

Кроме того целые числа тоже можно делить по разрядам и класть иногда только высшие разряды, чтобы также выбирать разряд для поиска.

5.3 Геопоиск

Бьем карту на кусочки для разных масштабов. Для кусочков используются широта и долгота, но нужно учитывать, что они кодируют разные расстояния в разных точках мира (на экваторе больше, у полюсов меньше).

Так же можно использовать KD-Trees.

Задание 5. Добавить поиск по числовым полям.

5.4 Сжатие индекса

Преимущества сжатия:

- Меньше требуется ресурсов на хранение.
- Быстрая загрузка данных с диска в память.

— Быстрая загрузка данных из памяти в кеш процессора.

Закон Хипса: $M \approx kT^b$, где

— M – количество различных терминов

— T – количество лексем в коллекции

— $b \approx 0.5$

— $30 \leqslant k \leqslant 100$

Закон Ципфа: $fn \approx \frac{1}{n}$, где fn – частота n-ого слова.

Если не сжимать для каждого слова нужно хранить 8-байтовый указатель, 2-байтовое число для количества символов и сами символы.

Для экономии места все строки объединяем в одну и храним только числовые позиции начала каждой из строк.

Можно сжать еще сильнее, храня блоками, храня начало блока, а в блоках размеры слов.

Кроме того, можно в начало блока записать общий префикс и дальше хранить только суффиксы.

В Lucene Для хранения используют конечный автомат: $a \to u \to t \to o \to End, a \to p \to t$

Задание 6. Посмотреть, как автомат работает в Lucene. Нужно засунуть наш словарь в этот автомат.

5.5 Сжатие инвертированного индекса

В индексе есть слово и id-шники документов, в которых оно встречается. Идшники могут быть большими, поэтому можно хранить не сами идшники, а разницы между ними.

Для этого используется кодирование переменной длины: в каждом байте храним 7 бит числа и один бит на флаг, закончилось ли число. На число может хватит одного или двух битов.

6 Ранжирование

6.1 Зональное ранжирование

Пример:

Книги:

— Автор
— Заголовок
— Аннотация
— Содержание

$\sum_i^n \alpha_i f_i$, где α_i – коэффициент зоны, f_i – значение зонной функции.

$\sum_j^k (\sum_i^n \alpha_i f_{ij} - r_j)^2 \to \min$

— Метод наименьших квадратов
— Квадратичная форма
— Градиентный спуск

Зональное ранжирование в Lucene:

— Document level boosting – выставляем веса документам
— Field level boosting – выставляем веса полям
— Query level boosting – в запросе задаем, у какого слова будет больший вес

6.2 Частота термина и взвешивание

— Bag of words
— Неважен порядок слов
— Пары документы и запрос
— Каждому слову в такой паре задается вес
— Основная характеристика – частота термина в документе.
— TF – сколько раз слово встретилось во всех документах.
— DF – сколько раз слово встретилось в конкретном документе.
— Не все слова одинаково полезны.
— Чтобы избежать этого словам присваивается частота

Обратная частота (idf): чем в большем числе документов встретился терм, тем менее он ценен:

$idf = \log \frac{N}{df_t}$

$tf \cdot idf_{t,d} = tf_{t,d} \cdot \log \frac{N}{df_t}$

Векторная модель ранжирования:

— Документ и слова представляем как вектора
— Косинусная мера похожести:
  $\cos\alpha = \frac{(d_i,d_j)}{||d_i||\cdot||d_j||}$ (в числителе скалярное произведение, потому что не нужно учитывать лишние слова?)
  $0 \leqslant \text{sim} \leqslant 1$
  Сублинейное масштабирование TF:

$$\begin{cases} 1 + \log tf_{t,d}, & \text{if } tf_{t,d} > 0 \\ 0, & \text{if } tf_{t,d} = 0 \end{cases}$$

  Нормировка tf на максимальный tf:
  $a + (1-a)\frac{tf_{t,d}}{\max tf_{t,d}}$
  Lucene ranking:

$$\text{coord} \cdot \text{queryNorm} \cdot \sum_{t \in q} \sqrt{tf_t} \cdot idf_t \cdot \text{boost}_t \cdot \text{norm}(t,d)$$

Формула BM25: одна из лучших форм для ранжирования, показывает очень хорошие результаты, для дальнейшего улучшения нужно применять уже более сложные методы и языковые модели.

$$\sum_{i=1}^{n} idf(q_i) \frac{tf(k_1+1)}{tf + k_1(1-b+b\frac{|D|}{\text{avgdl}})}$$

7 Оценка качества поиска

Необходимы элементы:

- Набор документов
- Набор запросов
- Набор оценок релевантности относительно каждой пары документ-запрос

Оценка неранжированных результатов поиска:

- Точность $P = \frac{tp}{tp+fp}$ – количество релевантных документов, что мы выдали на то, сколько всего мы выдали
- Полнота $R = \frac{tp}{tp+fn}$ – сколько релевантных документов мы выдали пользователю на то, сколько всего есть релевантных документов
- F-мера $F = \frac{2PR}{P+R}$ – среднее гармоническое для этих величин

В этой метрике не учитывается порядок, поэтому можно использовать метрику MAP (minimum average percision):

$$MAP(Q) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} \frac{1}{m_j} \sum_{k=1}^{m_j} P(R_{jk})$$

R_{jk} – результаты запроса с j по k

Метрика NDCG ():

$$NDCG(Q, k) = \frac{1}{|Q|} \sum_{j=1}^{|Q|} Z_k \sum_{m=1}^{k} \frac{2^{R(j,m)} - 1}{(1+m)}$$

$R(j, m)$ – релевантность m документа в j запросе

Z_k – подбираются таким образом, чтобы $NDCG$ была равна 1 при идеальной

Задание 7. Сделать assesment (для документ + запрос сделать оценки, от 0 до 2) и оценить качество поисковой системы. Использовать $NDCG$.

8 Вероятностная модель

Вероятностная модель ранжирования – считаем вероятность того, что документ релевантный.

В формулах d – document, q – query, R – является ли документ релевантным. Используется условная вероятность.

Сортировать можно не по полученным вероятностям, а по их отношениям.

Бинарная модель: учитываем вхождение слова только один раз, слова считаются независимыми.

Если в произведении нет $w_i = 0$ или $w_i = 1$ – оно не зависит от запроса.

df – document frequency, tf – term frequency, S – общее количество релевантных документов, N – общее количество документов, s – количество релевантных документов, в которых встретился данный термин из запроса.

9 Языковые модели

Вероятностная модель порождения языка – есть некоторая фраза, мы можем дописать туда какое-то слово.

Закрыв последнее слово фразы пытаемся угадать его. Лучше всего взять последние 4 слова и считать исходя из них.

Можем считать условные вероятности для каждого слова при всех предыдущих, можно учитывать условную вероятность только для последнего слова, а можно брать только вероятности сами слов.

$P(t_1, t_2, t_3, t_4) = P(t_1) \cdot P(t_2|t_1) \cdot P(t_3|t_2, t_1) \cdot P(t_4|t_1, t_2, t_3)$ – наиболее правильная модель, пытаемся ее упростить.

$P_{uni}(t_1, t_2, t_3, t_4) = P(t_1)P(t_2)P(t_3)P(t_4)$ – вероятностная модель (униграммная).

$P_{bi}(t_1, t_2, t_3, t_4) = P(t_1)P(t_2|t_1)P(t_3|t_2)P(t_4|t_3)$ – биграмная модель.

Задание 8. Построить вероятностную модель по нашему индексу. Униграммную (вообще без условной вероятности), двуграммную и трехграммную модель. Попробовать для каждой из этих моделей продолжить фразу.

Для языка существует много моделей. Языковая модель для документа, для корпуса.

$P(t|M_c)$ – языковая модель для корпуса.

$P(t|M_d)$ – языковая модель для документа.

$P(t|t_1)$

Модель правдоподобия запроса.

$P(d|q) = \frac{P(q|d)P(d)}{P(q)} \approx P(q|d)P(d)$

$P(d) = \frac{L_d!}{tf_1!...tf_n!}P(t_1)^{tf_1} \dots P(t_n)^{tf_n}$

$L_d = \sum_{i=1}^{n} tf_i$

$P(q|d) = P(q|M_d) = \frac{L_q!}{tf_{t_1,q}...tf_{t_k,q}} \prod P(t_k|M_d)^{tf_{t_k,q}}$

Данная модель не подходит, если термин из запроса не встречается в документе.

$P(t|M_d) = \lambda P_{MLE}(t|M_d) + (1 - \lambda)P_{MLE}(t|M_c)$

$P(t|M_d) = \frac{tf_{t,d} + \alpha P_{MLE}(t|M_c)}{L_d + \alpha}$

10 Современные языковые модели

Проблемы классического подхода: требуется большой объем оперативной памяти чтобы держать модель.

В корпус включаются только те последовательности слов, что встретились хотя бы 5 раз.

Языковая модель как рекуррентная нейронная сеть.

Слово представляется как вектор определенной размерности (от 50 до 300 в зависимости от задачи). Это удобно использовать как вход для нейронных сетей.

Word2Vec CBOW. По контексту нужно предсказать слово: word t-2, word t-1, word t+1, word t+2 $\rightarrow$ SUM $\rightarrow$ (softmax) word t.

Word2Vec Skip-gramm. По слову нужно предсказать контекст: наоборот из одного слова в 4.

Glove. По двум словам предсказать совместную встречаемость. word 1, word 2 $\rightarrow$ count

$$J = \sum_{i,j} f(x_{ij})(w_i^T w_j - \log X_{ij})^2$$

Задание 9. Обучить нейросеть Word2Vec, Glove или эмебдинг? на нашем корпусе товаров.

Простая ячейка. Вечная память или беспамятство – если сеть запомнила информацию, то она ее не забудет и из-за этого может на запомнить нужную информацию.

LSTM – long short term memory. Лишена проблемы затухания градиентов, память очищается.

RNN for text. Слово $\rightarrow$ embeding $\rightarrow$ LSTM $\rightarrow$ Out. LSTM $\rightarrow$ LSTM $\rightarrow$ . . . .

Bi-directional RNN for text. LSTM $\leftrightarrow$ LSTM $\leftrightarrow$ . . . . Может предсказывать слово в середине по контексту.

Depp netwroks. Слово $\rightarrow$ embeding $\rightarrow$ LSTM $\rightarrow$ LSTM $\rightarrow$ Out.

Stacked depp networks. В Out идут стрелки изо всех уровней LSTM.

Недостаток такой архитектуры: при переводе между языками порядок слов может меняться.

Механизм внимания позволяет избежать этой проблемы. Ищем H2 по выходам O1, O2, O3, O4 с различными весами. Для этого строится блок нейронной сети (linear softmax с 4 выходами (сумма равна 1)). С ее помощью

определяем веса w1, w2, w3, w4. H1 – выход нейронной сети для предыдущего слова. H3, H4 – будущие выходы для следующих слов.

Что еще можно посмотреть (рассматривать не будем):

- AWD-LSTM – ряд трюков, чтобы лучше натренировать модель.
- Transformer – другая архитектура, работает сразу с целыми предложениями.
- ULMFit – transfer learning для текстов на LSTM.
- BERT – transfer learning для текстов на Transformer.

11 Синонимия

Используется два поля – исходное поле и поле с синонимами. Полю с синонимами дают более низкий вес.

Синонимы добавляются с помощью специального фильтра.

Подходы для получения синонимов:

— Использовать уже готовые словари синонимов. Это работает не очень хорошо, поэтому обычно из словаря выкидывают то, что не нужно в данном домене.
— Автоматически строить словари.
— Латентно семантическое индексирование – устаревший способ, не рассматриваем.

Автоматическое построение синонимов. Какими свойствами должны обладать синонимы:

— Должны быть в одном и том же контексте.
— Они должны находится в одних и тех же документах.
— Если речь идет про веб, то в тексте ссылки может быть синоним на заголовок документа.

Используем контекст слова, обычно длина окна в 3-10 слов. Можно использовать Google ngrams.

Если просто брать топ-1000 – будет много просто популярных слов. Чтобы избежать этого используем меры схожести:

— tf-idf
— point wise mutual information – $PMI(word_1, word_2) = log_2 \frac{P(word_1, word_2)}{P(word_1)P(word_2)}$

  Если слова независимые, $PMI = 0$

  Другие способы получения синонимов:
— Транслитерация

  Вероятностная модель транслитерации:

  $P(word) = P(l_1)P(l_2|l_1) \dots P(l_n|l_1 \dots l_{n-1})$
— Словообразовательные расширения: Саратов – Саратовский
— Аббревиатуры
— Близость в пространстве вложений

12 Машинное обучение в ранжировании

Признаки для ранжирования на примере Google:

- Факторы домена:
  - Возраст домена – чем старше, тем луче
  - История домена – какие были сайты на этом домене
  - Привязка к стране
  - Публичный или приватный whois
  - Ключевое слово в домене
  - Ключевое слово в поддомене
  - Оштрафован ли владелец whois
  - Длительность регистрации домена
- Факторы оптимизации сайта
  - Ключевые слова в h1, h2, h3, h4, url, title, meta
  - Размер страниц
  - Уникальность наполнения
  - Исходящие ссылки
  - Видео и фото на сайте
  - Скорость загрузки
  - Битые ссылки
  - Правильность html
  - Частота обновления
- Факторы уровня сайта
  - Уникальность сайта
  - Обновления сайта
  - Наличие карты сайта
  - Место нахождение сервера
  - Наличие обратной связи
  - Количество страниц
  - Юзабилити сайта
  - Доступность сайта – как часто сайт находится не техобсулживании или подобном
  - Удобство навигации
- Обратные ссылки
  - Значение PageRank

  - Возраст ссылки
  - Количество ссылающихся страниц
  - Ссылки с доменов .edu .gov
  - Ссылки, добавленные пользователями
  - Качество ссылающего домена
  - Разнообразие типов ссылок
  - Позиция ссылки на странице
  - Слова в тексте ссылки и рядом стоящие слова
- Взаимодействие пользователей
  - Количество переходов по данному запросу
  - Количество переходов по всем запросам
  - Прямой трафик – переход из закладок или набирая адрес в ручную
  - Количество возвратов – как часто возвращается в поисковик после просмотра этого сайта
  - Почтовый трафик – как часто переходят с почтовых сервисов
  - Количество комментариев
  - Время проведенной на сайте

  Для большинства этих измерений нужно на сайте ставить специальные метрики, например Яндекс.Метрика или Гугл аналитику.
- Специальные правила
  - Запрос заслуживает свежести – для новостей нужны ответы последних пары дней
  - Запросы заслуживают разнообразия – например для просто запроса о холодильнике нужно выдать ему и магазины, и обзоры.
  - История запросов пользователя
  - Учет географии
  - Доменное разнообразие
  - Авторское право – ссылки, скрытые по запросу правообладателя
  - Подмешивание других вертикалей – картинки, новости
- Социальный сигналы
  - Количество твитов
  - Количество лайков на facebook, g+, vk
  - Авторитетность пользователя
  - Количество репостов, ретвитов

  - Релевантность аккаунтов
- Факторы спама
  - Фермы контента – сайты с кучей контента, созданные исключительно для денег с рекламы
  - Ссылки на плохие сайты
  - Всплывающая реклама
  - Реклама перед содержанием
  - Сгенерированный контент
  - Скрытые редиректы
- Факторы спама вне домена
  - Быстрый приток ссылок
  - Ссылки с одного и того же ip
  - Покупка и продажа ссылок
  - Ручной штраф
  - Релевантность ссылающего домена

Ранжирование для товаров:

- Сколько раз покупался товар
- Количество отзывов
- Количество положительных отзывов
- Разнообразие категории товаров

Learning to rank:

- Поточеченый подход
- Попарный подход
- Списочный подход

13 Learning to rank

— Поточечный подход – каждой паре число-документ нужно присвоить число, по которому будет сортировать.
— Попарный подход – тройка: два документа и запрос, для них определяем порядок, какой из документов лучше подходит. Подходит, когда мы не можем однозначно присвоить число.
— Списочный подход – получаем список документов и сортируем его. Практически не используется.

Нормирование параметров – перед тем, как заниматься ранжированием, нужно нормировать параметры. У параметров могут быть разные масштабы, поэтому некоторые параметры будут играть значительно большую роль.

$\frac{e^{ax}}{e^{ax}+1}$

Поточечный подход:

— Пара документ запрос – оценка релевантен или нет.
— Каждой паре соответствует набор признаков.
— Задача классификации $(d_i, q_i) = (x_{i1}, x_{i2}, \ldots, x_{in}) \to r_i$

Самый простой способ – линейная регрессия. Разбирали?

SVM классификация – строим прямую разделяющую, чем дальше от прямой – тем выше (ниже) ранг. $r_i = (\overline{w}, \overline{x}_i) + b$

Логистическая регрессия:

$P(r = 1|x_i) = \frac{e^{\sum \alpha_j x_{ij} + b}}{e^{\sum \alpha_j x_{ij} + b}}$

$P_{actual}(r = 1|x_i) \log P(r = 1|x_i) + (1 - P_{actual}(r = 1|x_i)) \log(1 - P(r = 1|x_i))$

13.1 Попарный подход

— Исходные данные – запрос q и пары документов (d1, d2).
— Для каждой пары документов известно, который подходит лучше.
— Нужно построить функцию ранжирования f(d).

Набор параметров и функция ранжирования:

$f(d_i) = f(x_1, x_2, \ldots, x_n)$

$f(d_i) = \sum_i^n \alpha_i x_i$

Модели:

— RankNet:
  – P_{ij} – вероятность, что документ d_i стоит выше, чем документ d_j
  – P_{actij} – наблюдаемая величина, принимает значения 0, 1

- $P_{ij} = \frac{\exp^{o_{ij}}}{\exp^{o_{ij}}+1}$, $o_{ij} = f(d_i) - f(d_j)$

  Нужно восстановить функцию ранжирования из функции потерь
- $L(D) = \sum_{q_k,d_i,d_j} -P_{actij} \log P_{ij} - (1 - P_{actij} \log(1 - P_{ij}))$
- Здесь очень важно использовать нормировку параметров.

— FRank:

- Функция потерь – $L(D) = \sum_{q_k,d_i,d_j} \left[1 - (\sqrt{P_{actij}P_{ij}} + \sqrt{(1 - P_{actij})(1 - P_{ij}}\right.$
- Перед использованием алгоритма делаем преклиппинг – максимальная и минимальная вероятность не 0 и 1, а например $(1e-5, 1-1e-5)$.

— Rank SVM:

- В оригинальном алгоритме SVM стараются сделать так, чтобы максимальный запрос был около прямой.
- У нас есть прямая, которая разделяет два множества – релевантные и нерелевантные.
- Для каждой точки вводим положительный штраф ξ_i, их сумму стараемся минимизировать.
- Функция потерь:

$$\begin{cases} \|w\| + C\sum \xi_i \to \min \\ y_i(wx_i - b) \leqslant 1 - \xi_i \\ \xi_i \geqslant 0 \end{cases}$$

- Затем решается какими-то хитрыми методами (не стохастическим градиентом, но можно и им, ибо проще).
- Функция потерь (вторая? в презентации разные картинки)

$$\begin{cases} ||w|| + C\sum \xi_{ij} \to \min \\ d_{ij}(w, x_i - x_j) \leqslant 1 - \xi_{ij} \\ \xi_{ij} \geqslant 0 \end{cases}$$

— SortNet

Недостатки попарного подхода:

— Следующие варианты ранжирования не различаются при попарном подходе (p – perfect, g – good, b – bad)

— g p g b b
— p g b g b
— Не учитывают количество документов на уровне запроса – есть 2 запроса, в одном нашлось много правильных документов, в другом документов вообще мало и все неправильные. Если используем попарный подход напрямую – получаем неправильную среднюю точность.
— Нужно дополнительно настраивать попарный подход, чтобы избежать этих проблем.

Это лечится с помощью приписывания веса запросу и паре документов – чем больше пар в запросе, тем должен быть меньше вес у каждой пары и запроса.

Припишем эти веса в нашу функцию потерь (ранжирования?). Проходим эпоху с ними, затем стохастическим градиентом обрабатываем эти веса.

Изменения в RankSVM: