# Big Data’13

## Лекция III: полнотекстовый поиск

Дмитрий Барашев
bigdata@barashev.net

Computer Science Center

7 марта 2013

# Сегодня в программе

Введение

Модели информационного поиска

Вычислительная часть

# Разминка

- У вас 8 машин, на каждой работает mapper и reducer
- У вас 800Мб пар (k, v) поделенные на 8 шардов
- k – целые числа, v – равномерно распределенные степени двойки от $2^0$ до $2^7$
- mapper разворачивает пару и выплевывает (v, k)
- reducer просто считает ключи
- каждый mapper и reducer за секунду может прочесть или записать 50Мб
- функция разбиения стандартная: хеш и взятие по модулю

# Разминка

- У вас 8 машин, на каждой работает mapper и reducer
- У вас 800Мб пар (k, v) поделенные на 8 шардов
- k – целые числа, v – равномерно распределенные степени двойки от $2^0$ до $2^7$
- mapper разворачивает пару и выплевывает (v, k)
- reducer просто считает ключи
- каждый mapper и reducer за секунду может прочесть или записать 50Мб
- функция разбиения стандартная: хеш и взятие по модулю

сколько секунд будет идти весь процесс и почему?
1) 6 секунд 2) 48 секунд 3) 14 секунд

# Сегодня в программе

Введение

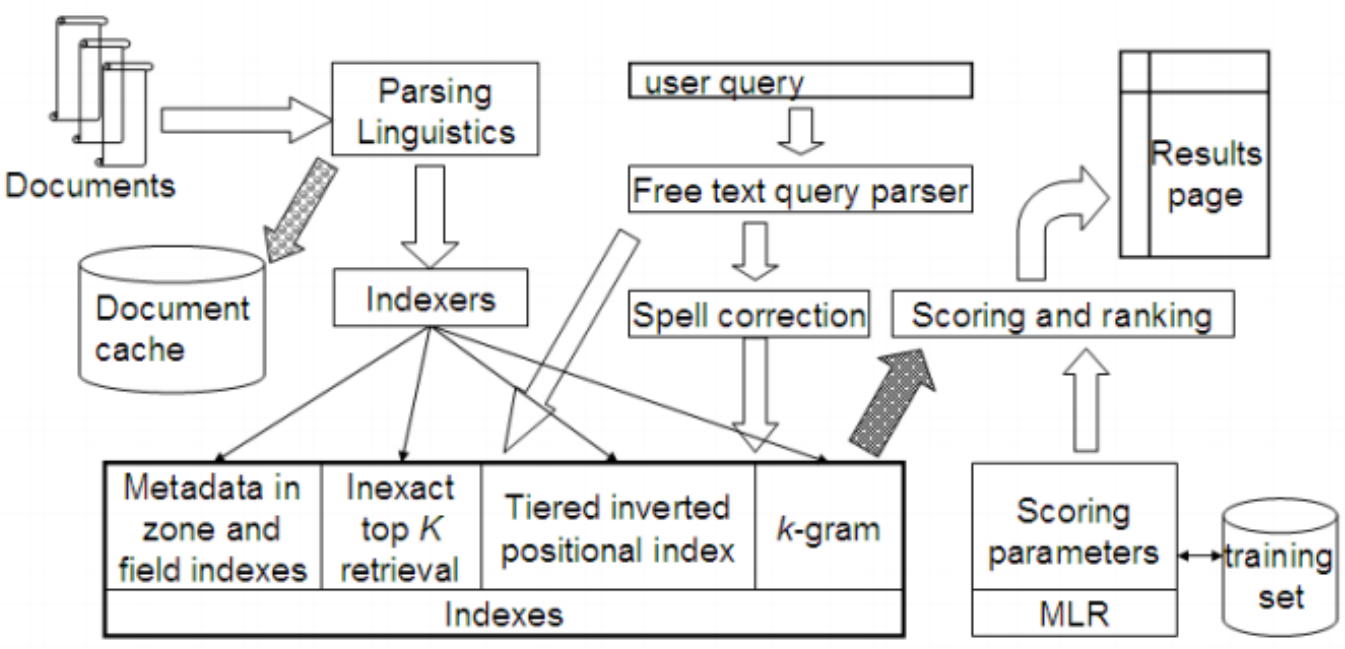
Модели информационного поиска

Вычислительная часть

# Задача

- Большой корпус текстовых документов
- Слабая структура или отсутствие структуры
- Неточные запросы

# Поисковая система



# Запрос и результат

- Как выглядит запрос?
- Каким должен быть результат?
- Что означает «документ удовлетворяет запросу» ?
- Все ли документы одинаково релевантны?
- Насколько точные запросы формулируют пользователи?

# Запрос и результат

- Как выглядит запрос?
- Каким должен быть результат?
- Что означает «документ удовлетворяет запросу» ?
- Все ли документы одинаково релевантны?
- Насколько точные запросы формулируют пользователи?

Ответы зависят от используемой модели поиска

# Качество поиска

- У пользователя есть *информационная потребность (information need)* сформированная запросом
- В зависимости от модели и реализации результаты могут быть разными
- Если в результатах много нерелевантных документов то поиск неточный
- Если много релевантных документов не попало в результат то поиск неполный

# Точность

$$P = \frac{|\{D_{rel}\} \cap \{D_{res}\}|}{|\{D_{res}\}|}$$

- Диапазон: $0 <= P <= 1$
- Точность равна 1 если в результате всего 1 документ и он релевантный

# Полнота

$$R = \frac{|\{D_{rel}\} \cap \{D_{res}\}|}{|\{D_{rel}\}|}$$

- Диапазон: $0 <= P <= 1$
- Полнота равна 1 если в результате все документы корпуса

# F-мера

$$F = 2 \times \frac{P * R}{P + R}$$

- Диапазон: $0 < F <= 1$
- F-мера равна 1 если все документы результата релевантны и других релевантных нет

# Сегодня в программе

Введение

Модели информационного поиска

Вычислительная часть

# Некоторые модели

- Поиск подстроки
- Поиск по регулярному выражению
- Булевская модель
- Векторная модель
- Вероятностная модель

# Подстрока и регулярные выражения

- Запрос – одна строка или одно регулярное выражение
  - ”`Computer Science Center`” или `(Computer\s+Science|CS)\s+Center`
- Документ рассматривается как одна большая строка или как список строк
- Документ, в котором нашлось совпадение подстроки/выражения, считается релевантным
- Обычно все документы в результате одинаково релевантны

# Булевский поиск

- Документ рассматривается как (мульти)множество слов
- Запросом являются слова, объединённые логическими операторами
- Слово в запросе заменяется на `true` если оно содержится в документе
- Релевантным называется документ, для которого полученное логическое выражение вернуло `true`

# Булевский поиск

- Документ рассматривается как (мульти)множество слов
- Запросом являются слова, объединённые логическими операторами
- Слово в запросе заменяется на `true` если оно содержится в документе
- Релевантным называется документ, для которого полученное логическое выражение вернуло `true`

Пример: `Барашев AND лекции AND NOT СПбГУ`

# Булевский поиск: недостатки

- Формулировка запросов – непростая задача
- Классическая модель бедна, расширенные сложны
- «Неустойчивость» результата к операторам
  - AND может сильно повысить точность но понизить полноту
  - OR наоборот, повысит полноту может существенно снизить точность
- Классическая модель не подразумевает ранжирования документов

# Булевский поиск: расширенные модели

- Дополнительные операции: «находится рядом», совпадение по подстроке
- Некоторое ранжирование: документ делится на «зоны» и зонам назначаются веса

# Векторный поиск: мотивация

- Пользователь может и не знать какие слова присутствуют в корпусе
  - *сколько нужно SEOшников чтобы вкрутить лампочку, лампочки, лампы накаливания, бра, светильники, доставка*
- Если слова в документе нет то это не повод документ отвергать

# Векторный поиск: мотивация

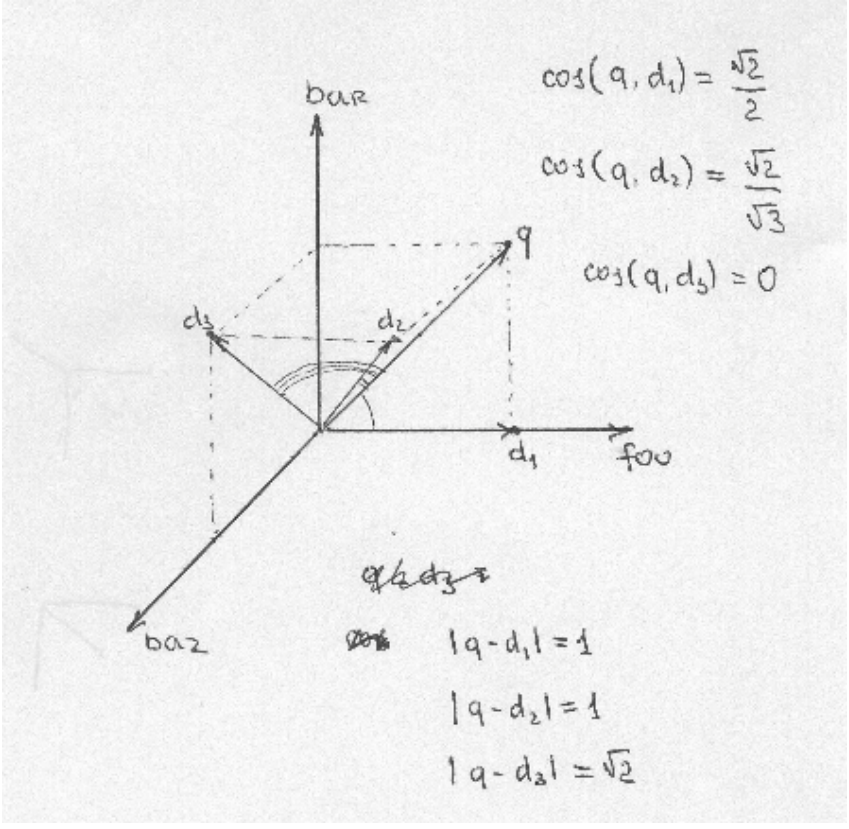
- Пользователь может и не знать какие слова присутствуют в корпусе
  - *сколько нужно SEOшников чтобы вкрутить лампочку, лампочки, лампы накаливания, бра, светильники, доставка*
- Если слова в документе нет то это не повод документ отвергать
- Не все слова одинаково ценны
  - *купить синхрофазотрон онлайн*

# Векторный поиск: абстрактная модель

- Документ $d$ и запрос $q$ – векторы в многомерном пространстве
- Измерениями являются слова, значениями координат вектора – некоторые числа, соответствующие паре $(d, w)$
- Релевантность документа запросу – какая-то мера близости двух векторов $sim(d, q)$

# Наивный пример

- Три измерения
- Координата равна 1 если слово есть в документе/запросе
- Мера близости – модуль разности векторов



# Локальная частота слова

- Слова в каждом документе неравноправны
- Если текст про конечные автоматы случайно содержит слово ”Калашников” то он будет равноправен с текстом, описывающим историю автомата Калашникова
- Давайте посчитаем $f_{t,d}$ – «сырую» частоту вхождения каждого слова в документ

# Локальная частота слова

- Слова в каждом документе неравноправны
- Если текст про конечные автоматы случайно содержит слово ”Калашников” то он будет равноправен с текстом, описывающим историю автомата Калашникова
- Давайте посчитаем $f_{t,d}$ – «сырую» частоту вхождения каждого слова в документ

теперь текст про автомат Калашникова скорее всего выиграет

# Нормализация сырой частоты

- Если просто считать число вхождений то более длинные документы будут выигрывать
- Сырую частоту можно нормализовать:
  - длиной документа:

$$tf_{t,d} = \frac{f_{t,d}}{|d|}$$

  - максимальной сырой частотой:

$$tf_{t,d} = \frac{f_{t,d}}{maxf_{w,d}, w \in d}$$

- или сделать ее рост нелинейным:

$$tf_{t,d} = 1 + logf_{t,d}$$

если $f_{t,d} > 0$

# Обратная встречаемость слова

- Слова вообще неравноправны
- *я, мы, что, где, когда* встречаются чуть менее чем везде, а вот *синхрофазотрон* еще надо поискать
  - кошмар гуглера: ВИА ”the the”
- Можно посчитать частоту слова во всем корпусе

# Обратная встречаемость слова

- Слова вообще неравноправны
- *я, мы, что, где, когда* встречаются чуть менее чем везде, а вот *синхрофазотрон* еще надо поискать
  - кошмар гуглера: ВИА ”the the”
- Можно посчитать частоту слова во всем корпусе
- Но более разумно:
  - посчитать сколько документов в корпусе содержат слово:

$$|d : t \in d|$$

# Обратная встречаемость слова

- Слова вообще неравноправны
- *я, мы, что, где, когда* встречаются чуть менее чем везде, а вот *синхрофазотрон* еще надо поискать
  - кошмар гуглера: ВИА ”the the”
- Можно посчитать частоту слова во всем корпусе
- Но более разумно:
  - посчитать сколько документов в корпусе содержат слово:

$$|d : t \in d|$$

  - и взять логарифм от обратной дроби:

$$idf_t = log \frac{|D|}{|d : t \in d|}$$

# TF-IDF

- Получили IDF - глобальную характеристику слова и TF - характеристику слова в конкретном документе

$$tf * idf_{t,d} = tf_{t,d} * idf_t$$

  - увеличивается когда вообще редкое слово часто встречается в одном документе
  - уменьшается когда слово встречается реже в документе или чаще в коллекции
  - равен нулю если слово присутствует во всех документах

# Варианты векторного пространства

- вектор документа $\overrightarrow{d}$ состоит из tf*idf, вектор запроса $\overrightarrow{q}$ из 1 или 0
- $\overrightarrow{d}$ состоит из tf, $\overrightarrow{q}$ из idf
- скалярное произведение $\overrightarrow{d} \cdot \overrightarrow{q}$ нормализуется эвклидовыми длинами:

$$sim(d,q) = \frac{\overrightarrow{d} \cdot \overrightarrow{q}}{|\overrightarrow{d}| * |\overrightarrow{q}|}$$

- или как-то иначе
- или рассматривается модуль расстояния между векторами: $sim(d,q) = |\overrightarrow{d} - \overrightarrow{q}|$

# Сегодня в программе

Введение

Модели информационного поиска

Вычислительная часть

# Эффективный поиск

- Линейно просмотреть все документы! $O(N)$!
- При 1млрд документов и 1мс на документ уложимся в 1млн секунд :)

# Эффективный поиск

- Линейно просмотреть все документы! $O(N)$!
- При 1млрд документов и 1мс на документ уложимся в 1млн секунд :)
- O(результата), а не O(корпуса) было бы лучше

# Матрица инцидентности

- Построим матрицу с терминами в строках и документами в столбцах

| | *compscicenter* | *amazon* | *microsoft* | *iStore* |
|---|---|---|---|---|
| *computer* | 1 | 1 | 0 | 1 |
| *science* | 1 | 0 | 0 | 0 |
| *center* | 1 | 0 | 1 | 1 |
| *research* | 0 | 0 | 1 | 0 |
| *book* | 0 | 1 | 0 | 0 |
| *sale* | 0 | 1 | 0 | 1 |

- булевский запрос `computer AND center AND NOT science` вычислится как `1101 AND 1011 AND 0111` и вернет `0001` – iStore

# Матрица инцидентности

- Построим матрицу с терминами в строках и документами в столбцах

| | *compscicenter* | *amazon* | *microsoft* | *iStore* |
|---|---|---|---|---|
| *computer* | 1 | 1 | 0 | 1 |
| *science* | 1 | 0 | 0 | 0 |
| *center* | 1 | 0 | 1 | 1 |
| *research* | 0 | 0 | 1 | 0 |
| *book* | 0 | 1 | 0 | 0 |
| *sale* | 0 | 1 | 0 | 1 |

- булевский запрос `computer AND center AND NOT science` вычислится как `1101 AND 1011 AND 0111` и вернет `0001` – iStore
- Битовая маска для 1млрд битов займет порядка 100Мб, но таких понадобятся сотни тысяч

# Инвертированный индекс

- Отображение терма в список содержащих его документов
- Документ идентифицируется численным идентификатором *docid*
- Множество термов называется *словарем (vocabulary, dictionary)*
- Списки называются *списками вхождений (posting lists)*
- Кроме *docid* элемент списка может содержать tf*idf и еще какую-нибудь информацию

# Построение индекса и подсчет tf*idf

- Map для каждого терма $t$ в документе $d$ выплевывает $(t, (docid_d, |d|))$
- Reduce может посчитать $tf_{t,d}$, записать список вхождений и, зная кол-во документов в корпусе, посчитать $idf_t$

# Булевский поиск с использованием индекса

- Пусть списки вхождений отсортированы по возрастанию *docid*
- Тогда запрос `computer AND center` можно вычислить алгоритмом похожим на *merge sort*

# Более сложные запросы

- В запросе несколько термов: `computer AND center AND NOT science`
- Можно воспользоваться *multiway merge sort*
- Можно составить деревянный план выполнения и начать оптимизировать:
  - сначала пересечь самые короткие списки
  - или положить самый маленький список в память и начать вычеркивать лишние документы

# А может SQL?

Запрос a AND b:

```
1 SELECT doc_id FROM PL JOIN T WHERE PL.term_id = T.id AND
      T.value='a'
2 INTERSECT
3 SELECT doc_id FROM PL JOIN T WHERE PL.term_id = T.id AND
      T.value='b'
```

# Векторный поиск с использованием индекса

- Общая мысль: нужны не все документы, а только самые релевантные
- Два подхода: пословный (term-at-a-time) и подокументный (document-at-a-time)

# Пословный поиск

```
1 scores = {}
2 for qt in query:
3   for dt in posting_lists[qt]:
4     scored[dt.doc] += qt.weight * dt.weight
5
6 for d in keys(scores):
7   scores[d] = scores[d] / d.length
8
9 return top_k(scores)
```

# Подокументный поиск

```
top_k = TopK()
plists = {}
for qt in query:
  plists[qt] = posting_lists[qt]
cur_doc_id = None
score = 0
while True:
  qt, dt = pull_min_doc_id(plists)
  if dt.doc_id != cur_doc_id:
    top_k[cur_doc_id] = score
    score = 0
    cur_doc_id = dt.doc_id
  if cur_doc_id is None:
    break
  score += qt.weight * dt.weight
```

# Упорядочивание списков вхождений

- Нужно любое, одинаковое во всех списках
- Возрастание docid – окей, но можно ли лучше?
- Например чтобы самые «ценные» документы были в начале списка

# Статический ранг документа

- Некоторая характеристика, не зависящая от запроса
- Выше ранг – ценнее документ
- Упорядочивание по статическому рангу позволит закончить поиск быстрее

# Другие способы сократить поиск

- *Чемпионский список* для терма $t$: документы с весом $tf_t$ выше какого-то порогового значения или просто top-m документов в порядке убывания веса терма
- Слоеный индекс, делящий документы по статическому рангу, «свежести», и т.д.

Эта презентация сверстана в

PAPEERIA

LaTeX в вашем браузере

alpha.papeeria.com

# Литература I


Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze.
*Introduction to information retrieval*, volume 1.
Cambridge University Press Cambridge, 2008.