Home

Информационный поиск

Лекция 2. Tf-Idf и BM25

Telegram

GitHub

Ника Зыкова
2023/09/06

С чего начинаем?

У нас все еще есть корпус, состоящий из нескольких текстов:

doc_1 = Буря мглою небо кроет

doc_2 = Вихри снежные крутя

doc_3 = То, как зверь, она завоет

doc_4 = То заплачет, как дитя

И прямой индекс для него.

Документ	Списко слов
doc_1	буря, кроет, мглою, небо
doc_2	вихри, крутя, снежные
doc_3	завоет, зверь, как, она, то
doc_4	дитя, заплачет, как, то

Обратный индекс

Но с прямым работать не так удобно, поэтому будем использовать обратный. Сначала в виде словаря, потом в виде матрицы. Обычно хранят еще частоту встречаемости, но в нашем случае это не актуально.

```
{
        "буря": [
                "doc_1"
        ],
        "то": [
                "doc_3",
                "doc_4"
        ],
        "как": [
                "doc_3",
                "doc_4"
        ],
  ...
}
```

	буря	мглою	небо	кроет	вихри	снежные	крутя	...
doc_1	1	1	1	1	0	0	0	
doc_2	0	0	0	0	1	1	1	
doc_3	0	0	0	0	0	0	0	
doc_4	0	0	0	0	0	0	0	

Входные данные

Корпус = {

doc_1: Буря мглою небо кроет,

doc_2: Вихри снежные крутя,

doc_3: То, как зверь, она завоет,

doc_4: То заплачет, как дитя

}

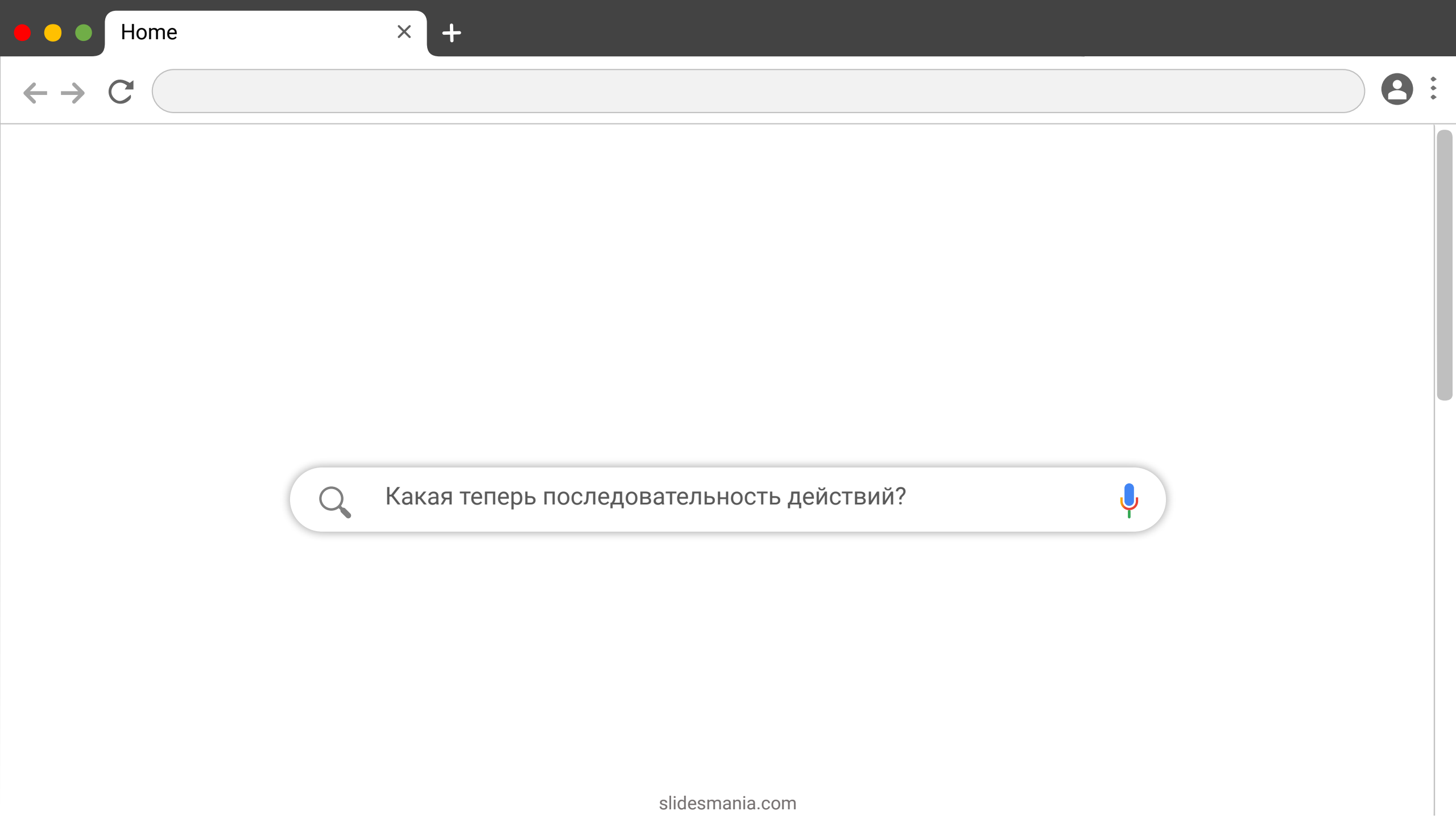
Запрос = "буря заплачет над небом, над небом"

Метрика = количество общих слов в запросе и документе

Задача: найти документ, больше всего подходящий под запрос

Индекс по корпусу:

```
{
        "буря": [
                "doc_1"
        ],
        "то": [
                "doc_3",
                "doc_4"
        ],
        "как": [
                "doc_3",
                "doc_4"
        ],
  ...
}
```



Home | Recent Tabs | Current Page

You can type something here...

Порядок поиска

Что нужно получить: для каждого документа нужно подсчитать степень его релевантности запросу, потом отсортировать документу в соответствии с полученными значениями.

Для этого:

1. идем по документам в корпусе
2. считаем метрику для пары (запрос, документ)
3. сохраняем полученное значение
4. сортируем документы по значению метрики

В чем здесь проблема?

Используем преимущества индекса

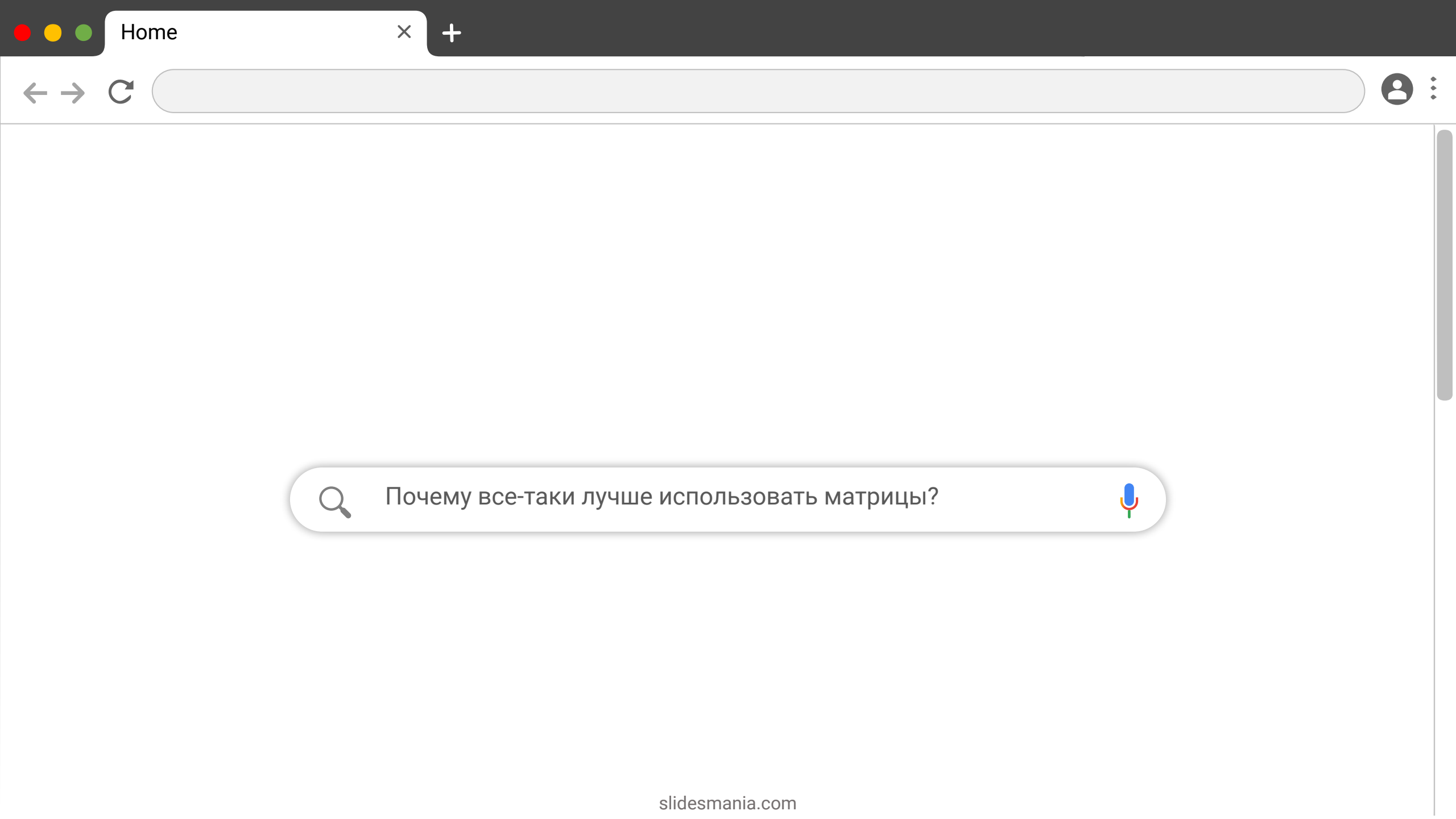
В предыдущем варианте поиска мы никак не учитывали тот факт, что у нас есть обратный индекс (с которым описанный алгоритм работает очень плохо).

В реальной ситуации у нас тысячи документов и сотни тысяч слов. Хотим ли мы перебирать все?

Новый алгоритм:

1. идем по уникальным словам в запросе
2. для документов, где встретилось слово, плюсуем метрику
3. финализируем подсчет метрики (например, нормализуем ее)
4. сохраняем полученное значение
5. сортируем документы по значению метрики

Почему такой алгоритм выгоднее?



Почему матрица?

Это даст нам:

- более простое и лаконичное решение за счет перехода от алгоритма к математике (это не всегда так работает, но часто)
- буст по скорости поиска во много раз за счет скорости операций над матрицами

А какой все-таки у нас индекс: прямой или обратный?

	буря	мглою	небо	кроет	вихри	снежные	крутя	...
doc_1	1	1	1	1	0	0	0	
doc_2	0	0	0	0	1	1	1	
doc_3	0	0	0	0	0	0	0	
doc_4	0	0	0	0	0	0	0	

Легким движением руки...

Прямой индекс превращается в обратный и наоборот. Ведь если мы говорим о матрице, то это изменение - лишь вопрос транспонирования.

Еще часто говорят о document-term (DT) и term-document (TD) матрицах, где первое слово обозначает информацию в строках, а второе - в столбцах.

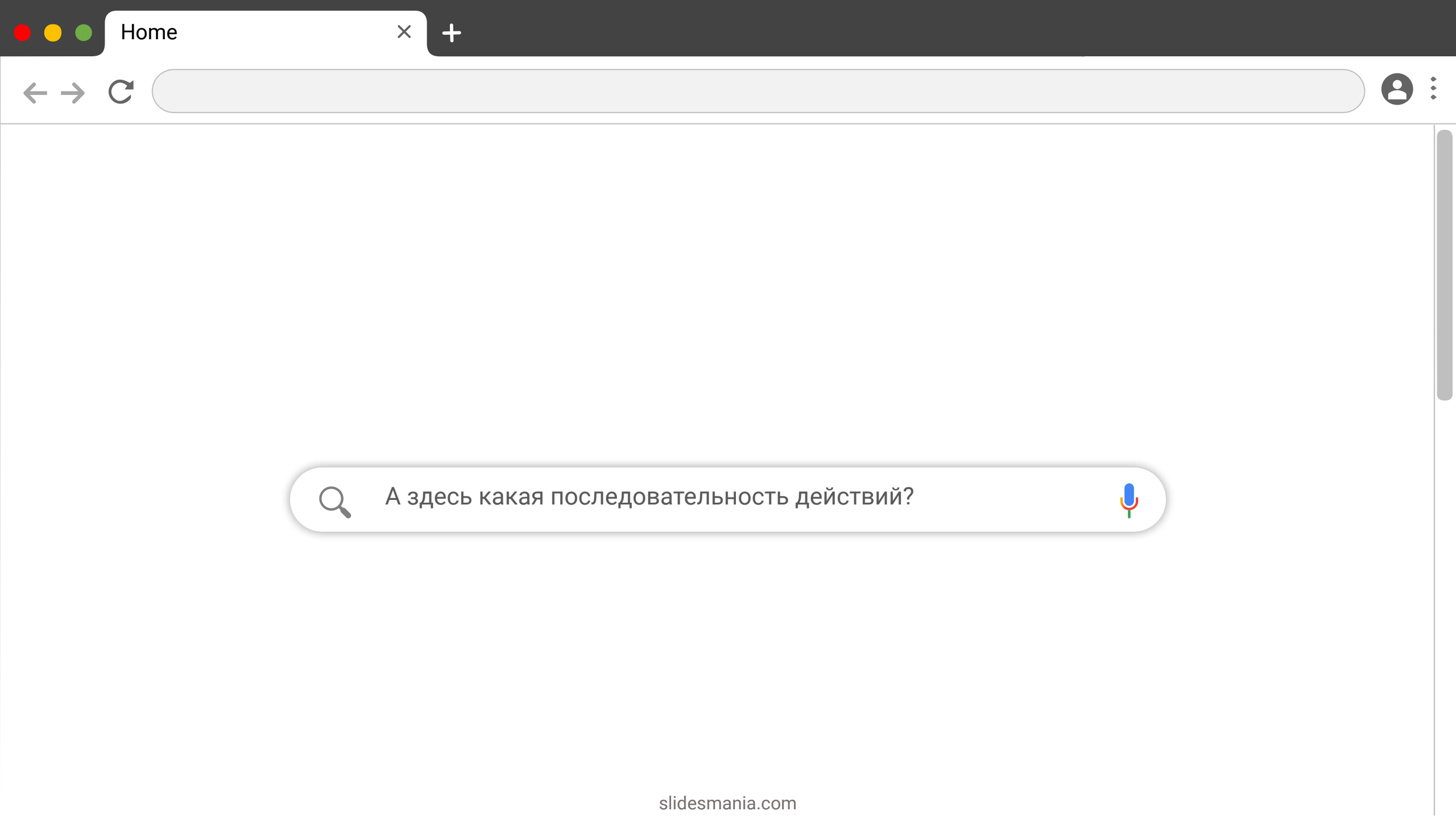
	буря	мглою	небо	кроет	вихри	снежные	крутя	...
doc_1	1	1	1	1	0	0	0	
doc_2	0	0	0	0	1	1	1	
doc_3	0	0	0	0	0	0	0	
doc_4	0	0	0	0	0	0	0	

Что может быть в ячейках?

Значения элементов матрицы могут быть различными, например:

- бинарный показатель есть / нет в документе
- какой-то статистический показатель, например, количество в документе
- TF-IDF или BM-25
- и другие показатели значимости слова в документе

	буря	мглою	небо	кроет	вихри	снежные	крутя	...
doc_1	1	1	1	1	0	0	0	
doc_2	0	0	0	0	1	1	1	
doc_3	0	0	0	0	0	0	0	
doc_4	0	0	0	0	0	0	0	



Порядок поиска с матрицей

Нужно все то же самое: оценка релевантности каждого документа запросу.

Для этого:

1. превращаем запрос в вектор тем же способом, что и документы (в нашем случае - считаем частоту слов)
2. умножаем DC матрицу на вектор запроса, получаем метрики сразу для всех документов
3. сохраняем полученное значение
4. сортируем документы по значению метрики

А в чем здесь может быть проблема?

Сравнение

	СЛОВАРЬ	МАТРИЦА
Простота обработки	0	1
Скорость поиска	0	1
Прозрачность логики	0	1

Только это не совсем честная табличка. Почему?

Сравнение (если честно)

	СЛОВАРЬ	МАТРИЦА
Простота обработки	0	1
Скорость поиска	0	1
Прозрачность логики	0	1
Разреженные данные	1	0
Человекочитаемость	1	0

А вот теперь все не так однозначно. Вообще, на практике чаще всего используют вариант матриц, где учтена проблема с разреженными данными: sparse матрицы.

Промежуточное итого

Наша идея поиска – умножение матрицы на вектор для получения близости между запросом и документами из корпуса

Матрица – это проиндексированная коллекция документов

Вектор – это проиндексированный запрос

Результат - ранжированные по убыванию релевантности запросу документы

Home | Recent Tabs | Current Page

You can type something here...

Векторизация документов

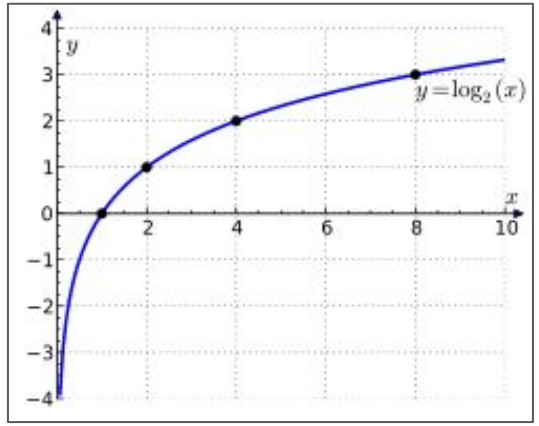
Как уже было сказано, способов множество:

- бинарное кодирование (есть/нет) - OneHotEncoder (sklearn, но так никто не делает)
- кодирование по частоте - CountVectorizer (sklearn)
- TF-IDF - TfidfVectorizer (sklearn)
- BM25 - библиотека rank-bm25
- word2vec - gensim (чаще всего)
- bert - pytorch (чаще всего)
- и тд

Вспомним Tf-Idf

$$Metric_{x,y} = tf_{x,y} \times \log(\frac{N}{df_x})$$

где x - слово, y - документ



- В чем общая идея?
- Какие части и что означают?
- Почему логарифм?

Формула BM25

$$bm25(Query, Doc) = \sum_{i=1}^{n} IDF \cdot \frac{TF \cdot (k+1)}{TF + k \cdot (1 - b + b \cdot \frac{l(d)}{avgdl})}$$

Объясним каждую компоненту:

- n - количество слов в запросе
- IDF - обратная документная частота слова q_i из запроса Query
- TF - частота слова q_i в документе Doc
- k - магическая константа
- b - магическая константа
- l(d) - количество слов в документе Doc
- avgdl - константа = средняя длина документа в корпусе

Части формулы

Все, что не зависит от запроса, нужно посчитать заранее. Это даст более оптимальное время исполнения поиска.

Не зависит от запроса:

- N – кол-во документов в корпусе
- k = 2
- b = 0.75
- l(d) – кол-во слов в документе Doc
- avgdl – средняя длина документа в корпусе

Зависит от запроса:

- TF(q_i, Doc) – частота q_i в Doc
- n(q_i) – кол-во доков, где есть q_i

Вопросы на подумать

- Почему обратный индекс, а не прямой?
- Какие есть плюсы и минусы у каждого типа поиска (булев, сходство, релевантность)?
- Из каких этапов состоит разработка поисковика? Что нужно сделать с текстами?
- Какие вы можете придумать способы индексирования разных типов данных?
- Чем принципиально BM25 отличается от Tf-Idf?
- Что дают константы в формуле?