

Текстовое ранжирование I

Информационный поиск. Лекция №5



Андрей Кривой

О преподавателях



Андрей Кривой

Руководитель группы ранжирования.

Занимаюсь качеством поиска в поисковых проектах VK

Telegram: @mt6qmzaotzn3

Лекции

Часть I: инженерная

1. Введение
2. Поисковый робот
3. Индексация и булев поиск
4. Предобработка запросов. Исправление опечаток

Часть III: нейропоиск

9. Нейропоиск: семантический матчинг
10. Нейропоиск: переранжирование
11. Нейропоиск: векторный поиск
12. Мультимедийный поиск

Часть II: ранжирование

5. Текстовое ранжирование I: метрики и векторные модели
6. Текстовое ранжирование II: вероятностные модели
7. Поведенческое и ссылочное ранжирование
8. Машинное обучение ранжированию

Сегодня мы тут!



О чем будем говорить

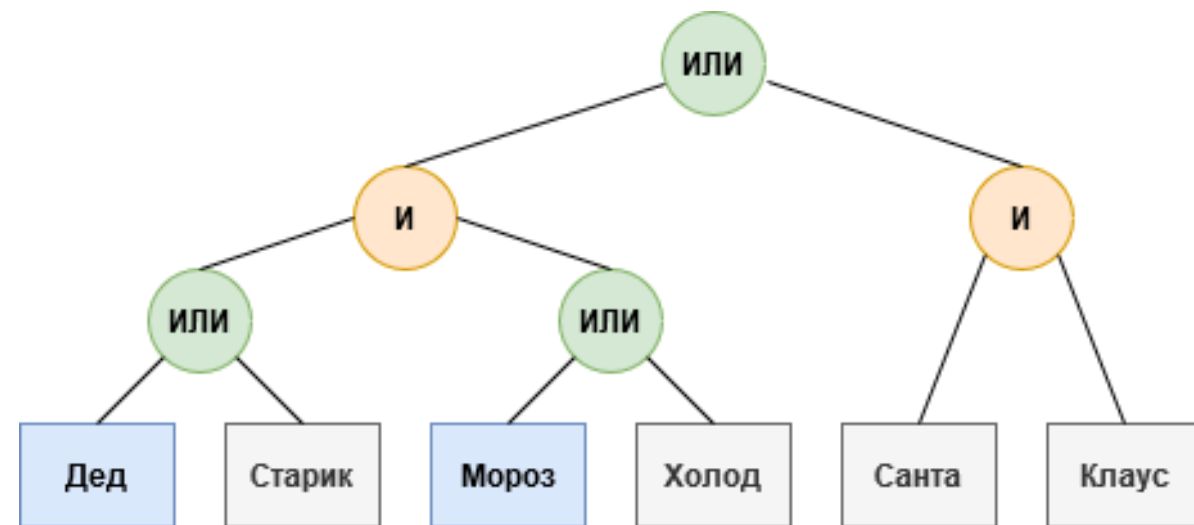
1. Вспомним задачу ранжирования
2. «Наивное» текстовое ранжирование, TF-IDF
3. Текстовое ранжирование на практике
4. Модель векторного пространства (Vector Space Model)
5. Латентно-семантический анализ (LSA)
6. Семинар и ДЗ

Задача ранжирования



Булев поиск

- Вспомним булеву модель поиска (Boolean retrieval model)
- Моделируем многословные запросы с помощью логических операций
- Поддерживаем И- и ИЛИ-запросы, а также их комбинации произвольной вложенности
- Пример: *((дед || старик) && (мороз || холод)) || (санта && клаус))*
- В общем случае такой запрос можно представить в виде дерева: узлы-операции и листья-термины
- Сливаем и пересекаем найденные списки документов двигаясь снизу вверх по дереву
- Красивая и строгая семантика!



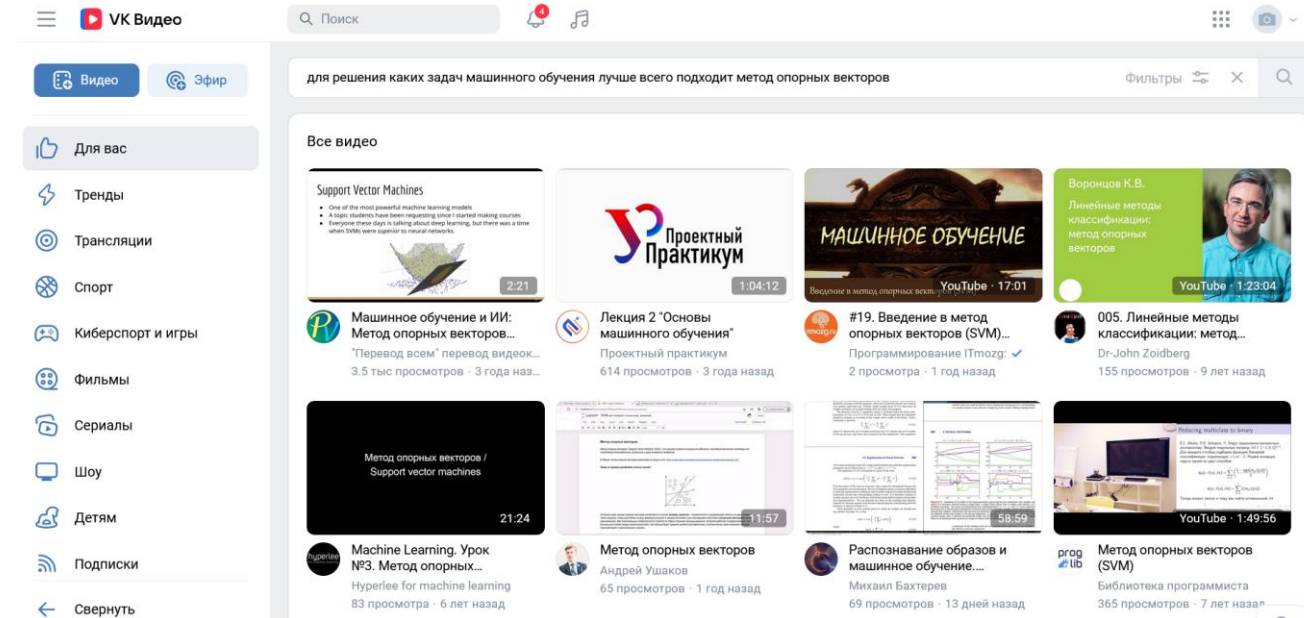
Сложный булев запрос представленный в виде дерева

Использованы синонимы:

- ДЕД МОРОЗ == САНТА КЛАУС
- ДЕД == СТАРИК
- МОРОЗ == ХОЛОД

Ad hoc* поиск

- У булева поиска есть серьезные проблемы!
- И-запросы обрабатываются слишком жестко: мы хотим поднимать документы, в которых есть только часть слов запроса
- Пользователи не могут проглядывать глазами миллионы найденных документов, содержащих ключевые слова – нужно ранжирование!
- Нужен т.н. *ad hoc* поиск: обработка запросов в свободной форме + ранжирование результатов



Обработка очень многословного *ad hoc* запроса «для решения каких задач машинного обучения лучше всего подходит метод опорных векторов»

Этот запрос, очевидно, нельзя обрабатывать как «строгий» И-запрос!

***Ad hoc** — латинская фраза, означающая «для данного случая», «специально для этого».

Ранжирование

- Основная идея: хотим, чтобы релевантные запросу документы стояли в поисковой выдаче выше, чем нерелевантные
- На практике это достигается за счет того, что каждому документу D по запросу Q назначается некий ранк (он же скор, score) $Score(q, d)$, по которому документы можно ранжировать (сортировать в порядке убывания ранка)
- ВНИМАНИЕ: ранк зависит от запроса!

Запрос «что лучше всего есть вечером»

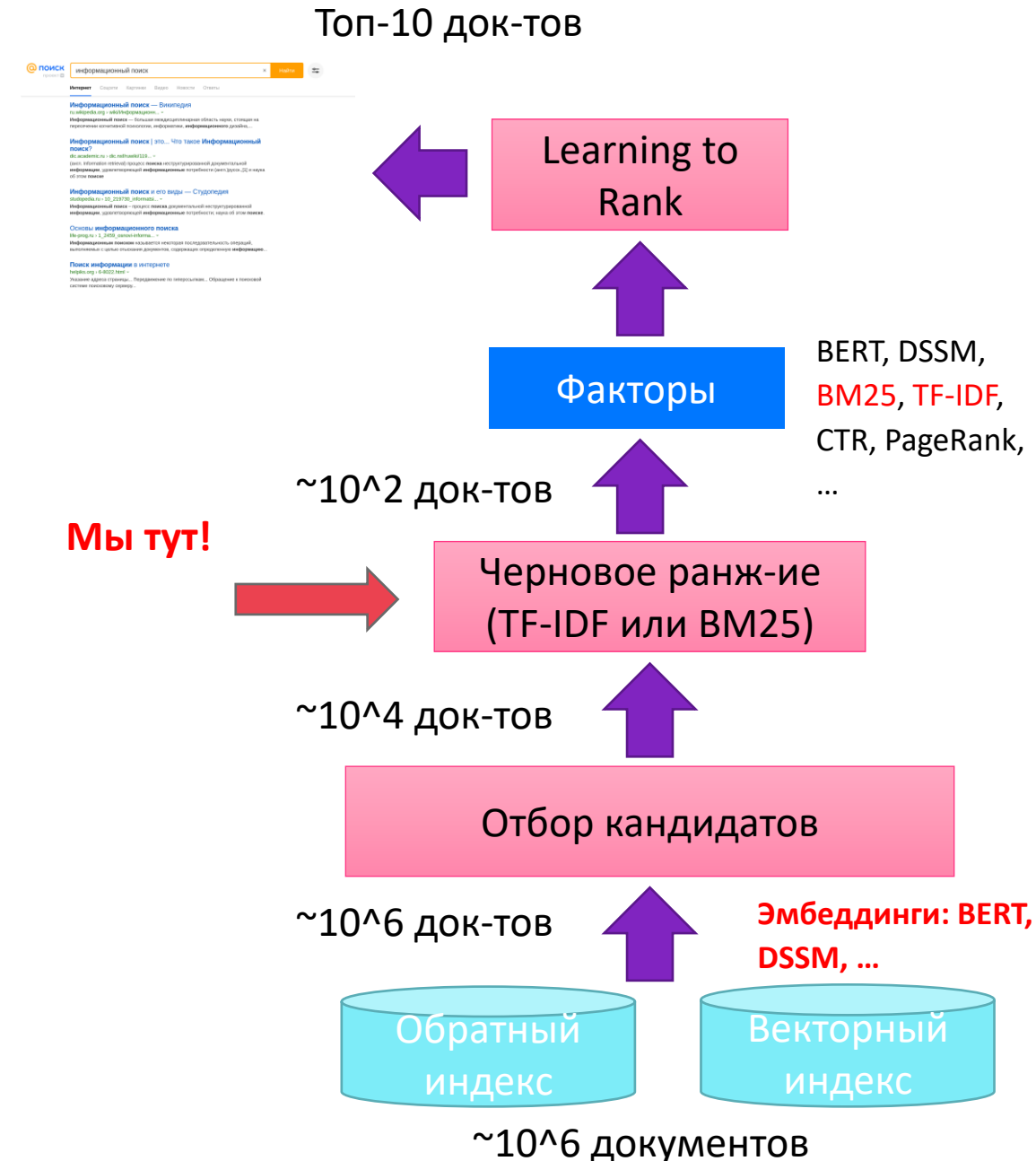
[illegible]

Ранжирование – это просто?

Пример из тестовой выдачи поиска по постам Дзена (dzen.ru)

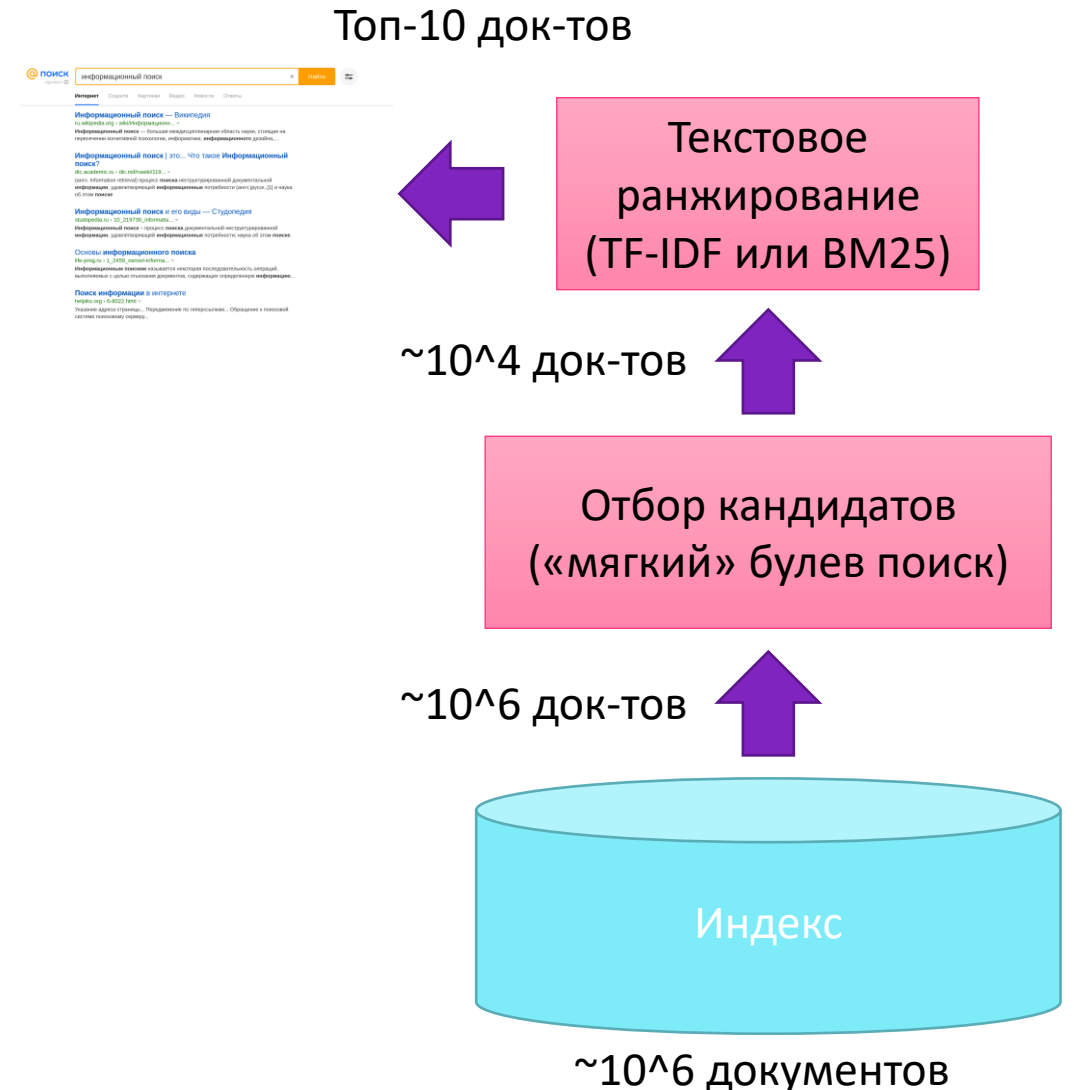
Пирамида ранжирования

- Современная поисковая система выглядит как-то так
- На этапе отбора кандидатов мы оставляем только документы, в которых найдены ключевые слова (булев поиск с «мягким» И)
- На этапе черного ранжирования мы применяем простые алгоритмы текстового ранжирования (*TF-IDF* или *BM25*) и отбираем топ-К кандидатов
- Вычисляем для каждого из топ-К кандидатов ~ 1000 факторов: нейрофики (*BERT*, *DSSM*, ...), текстовые фики (*TF-IDF*, *BM25*, ...), поведенческие фики (*CTR*, ...) и др.
- «Замешиваем» факторы в LTR-модель
- Используем ранки, которые выдает LTR-модель, для формирования финальной выдачи



Только текстовое ранжирование

- При желании, можно обойтись только текстовым ранжированием
- В таком случае поисковая выдача – это документы, просто сортированные по их *TF-IDF* и *BM25*
- Дает хорошее качество поиска «из коробки», без какого-либо ML
- Так работали все поисковики до начала 90-х (т.е. до появления web-поиска)
- Именно работает поисковый движок **Whoosh**
- В Поиске VK до 2012 г. в бою работала ручная формула, в которой креативно смешивались *BM25* и поведенческие факторы (такие как *CTR*)



Первая цель на сегодня

- Хотим научиться вычислять текстовые ранки, такие как *TF-IDF*
- Дальше эти ранки можно применять как:
 - Сами по себе, для чернового ранжирования (в современном поиске) или финального ранжирования (в простом поиске без ML)
 - Как факторы, которые замешиваются в более высокоуровневую модель ранжирования (LTR, Learning to Rank)
- Текстовое ранжирование до сих пор важно и нужно!

«Наивное» текстовое ранжирование



Постановка задачи

- У нас есть корпус из текстовых документов
- Пользователь подает поисковые запросы
- Для каждой пары запрос-документ надо получить такой ранк (score), чтобы при сортировке документов по этому ранку оптимизировались метрики качества поиска, такие как precision или recall
- У нас есть только тексты запросов и документов => наше решение должно опираться только на них
- Такое ранжирование называют *текстовым ранжированием*
- Тут, конечно, тоже можно применить ML, но мы начнем с чего-то более простого
- В первой части лекции мы будем рассматривать только аналитические решения (*классическое текстовое ранжирование*)
- А во второй части лекции у нас появится что-то похожее на ML

TF

- Рассмотрим сначала однословные запросы, напр. «физтех»
 - В корпусе могут быть миллионы документов которые содержат слово (термин) ФИЗТЕХ
 - Определим TF (term frequency, частота термина) – сколько раз данный термин встречается в документе
 - Разумное предположение: чем чаще термин запроса встречается в документе, тем лучше!
 - Будем просто ранжировать выдачу по однословным запросам в порядке убывания TF
- Запрос «*физтех*»
 - Какой из документов релевантнее?
 1. *Страница википедии про МФТИ ([ссылка](#))* – термин ФИЗТЕХ встречается на ней 107 раз!
 2. *Страница википедии про город Долгопрудный ([ссылка](#))* – термин ФИЗТЕХ встречается на ней всего 3 раза

Многословные запросы

- Распространим идею ранжирования с помощью TF на многословные запросы
- Вычислим для каждой пары из запроса q и документа d ранк $TfScore$ (его еще называют *overlap score*):
- $TfScore(q, d) = \sum_{t_i \in q} TF(t_i, d)$ где:
 - Суммируем по всем терминам (словам) t_i запроса q
 - $TF(t_i, d)$ – частота встречаемости t_i в документе d
- Для каждого запроса q , ранжируем поднятые из индекса документы по их $TfScore(q, d)$
- Получим не очень качественное, но все же ранжирование (и оно будет уже лучше рандома!)

Welcome to Red Apples Online!

If red apples are what you're looking for, then you are definitely in the right place to buy red apples. When it comes to red apples, you won't find a higher quality selection of red apples anywhere! Our red apples experts know how to pick only the best, most savory red apples from the bunch, and we sell these premium red apples right here for you to enjoy (red apples). Seriously, go to another red apples website and try their red apples. We guarantee you'll come crawling back to buy our red apples, buddy.

Пример как спамеры, к сожалению, могут легко заабыюзить наш $TfScore$

(такой спам называется *keyword stuffing*)

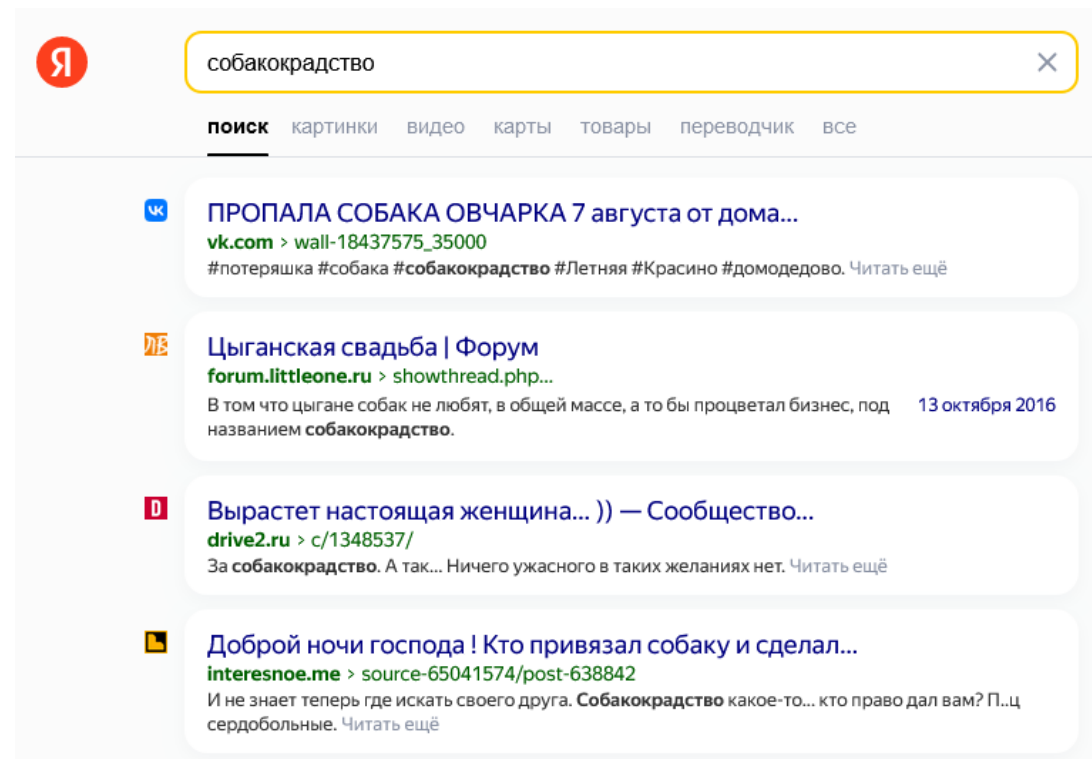
Важность слов

- Проблема: не все слова одинаково полезны
 - Запрос «институт итмо»
 - Документы, в которых есть только слово ИТМО очевидно гораздо полезнее документов в которых есть только слово ИНСТИТУТ
 - Нужна как-то учитывать относительную важность слов!
- Какой из документов релевантнее по запросу «институт итмо»?
 1. *Московский физико-технический институт*
 2. *Университет **ИТМО***

У этих документов одинаковый *TfScore*, но, очевидно, 2-й документ более релевантен

Документная частота

- Наблюдение: информативность термина зависит от его частоты
- Чем реже встречается термин – тем более он информативен и, как следствие, важен для поиска
- Например, в запросе «административная ответственность за собакокрадство», очевидно, заведомо самым важным является термин СОБАКОРАДСТВО
- Обратный пример – это т.н. стоп-слова типа (И, В, НА и т.п.). Они несут очень мало информации и часто просто игнорируются поисковыми системами
- Кажется, что хорошей мерой информативности термина может стать его *документная частота* (*document frequency*, или просто *DF*) – число документов, в которых встречается данный термин



Яндекс находит всего 4 документа со словом
СОБАКОКРАДСТВО

(актуально на 04.2024)

IDF

- На практике обычно используют не обычную документную частоту, а *обратную документную частоту (inverted document frequency, IDF)*
- Пусть: N – это полное число документов в корпусе, а N_t – это число документов в корпусе, в которых содержится термин t
- Определим: $IDF(t) = \log \frac{N}{N_t}$
- Почему именно логарифм? Выведем это позже!
- Проблема: деление на 0 в случае когда $N_t = 0$
- Поможет сглаживание: $IDF(t) = \log \frac{N}{N_t + 1} + 1$

- Пример расчета IDF на корпусе из 3х документов:
 1. *Московский физико-технический институт*
 2. *Московский государственный университет*
 3. *Университет ИТМО*

$$IDF(\text{ИТМО}) = \log(3/1) = 1.099$$

$$IDF(\text{МОСКОВСКИЙ}) = \log(3/2) = 0.405$$

$$IDF(\text{УНИВЕРСИТЕТ}) = \log(3/2) = 0.405$$

... и т.д.

Ранжируем с помощью IDF

- Определим $IdfScore(q, d) = \sum_{t_i \in q \ \& \ t_i \in d} IDF(t_i)$ где
 - суммируем только по тем терминам t_i из запроса q которые есть в документе d
- ВНИМАНИЕ: $IDF(t)$ – это *глобальная* статистика, которая считается по всему корпусу, и не зависит от документа d , который мы сейчас ранжируем
- => по однословным запросам у всех документов будут одинаковые $IdfScore(q, d)$
- Теперь многословные запросы ранжируются хорошо, а однословные – плохо!

TF-IDF

- Что если все объединить?
- $TfIdfScore(q, d) = \sum_{t_i \in q} TF(t_i, d) * IDF(t_i)$, где сумма по всем словам запроса
- Будем ранжировать документы по их $TfIdfScore(q, d)$
- Лучшее из обоих миров: хорошо работает и для однословных, и для многословных запросов
- Объединяет локальную (TF) и глобальную (IDF) статистику
- Формула TF-IDF появилась в 70-е годы и до сих пор так или иначе используется каждой поисковой системой
- Широко используются не только в поиске, но и в других задачах NLP



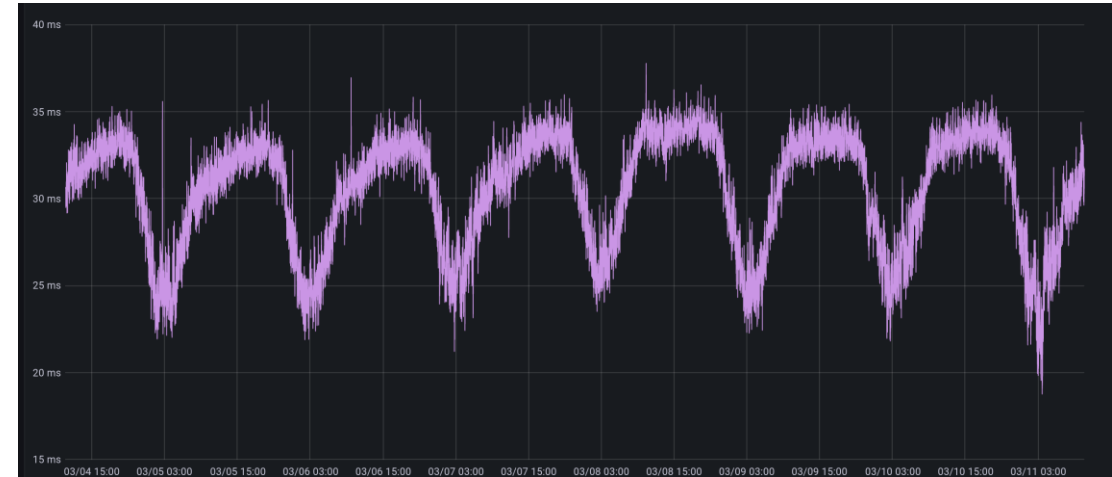
Karen Spärck Jones (26 August 1935 – 4 April 2007) – ученый Кембриджского Университета, которая в начале 70-х годов придумала IDF ([википедия](#))

Текстовое ранжирование на практике



Как применить TF-IDF?

- Корпус из 1 млрд. документов (типично для проектов масштаба VK)
- Наивный подход: считаем для *TfidfScore* каждой пары запрос-документ
- Частота CPU $\sim 10^9$ GHz \Rightarrow ранжирование 1 запроса займет заведомо больше > 1 сек (на самом деле гораздо дольше)
- Надо как-то оптимизировать



Среднее время ответа бэкенда поиска VK Видео.

Даже днем (в час пик) бэкенд успевает отвечать за **35 мс**, за это время он успевает:

- найти документы в обратном индексе
- рассчитать ~ 1000 фичей
- применить модель ранжирования

TF-IDF + обратный индекс

- Ключевая идея: $TfidfScore(q, d) = 0$ если в документе нет ключевых слов запроса
- Значит, можно воспользоваться обратным индексом
- В среднем для какого-то конкретного запроса число документов с ключевыми словами значительно меньше полного числа документов
- => сокращаем число документов-кандидатов для которых надо посчитать ранки в 100 и более раз

Обратный индекс

- *государственный* → [2]
- *институт* → [1]
- *итмо* → [3]
- *московский* → [1, 2]
- *технический* → [1]
- *физика* → [1]
- *университет* → [2, 3]

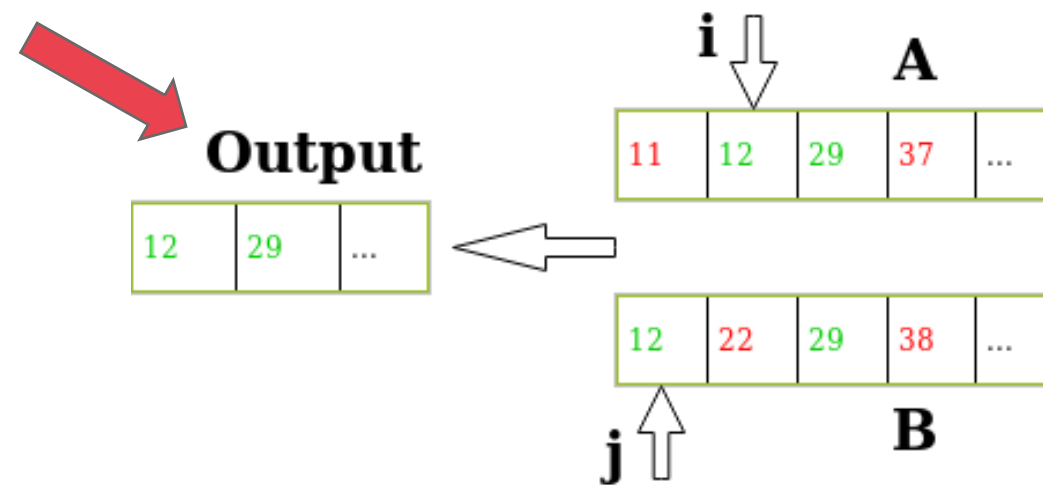
Как выглядит обратный индекс.

С его помощью мы легко можем найти документы в которых содержится ключевое слово.

Отбор кандидатов

- Попытка #1: моделируем все запросы как И-запросы и отбираем кандидатов с помощью булева поиска
- Считаем TF-IDF и другие ранки только для тех документов, в которых нашлись все слова запроса

Считаем
 $\text{TfIdfScore}(q,d)$
только для них!



Алгоритм пересечения 2-х сортированных списков документов, которые были найдены по ключевым словам в обратном индексе

Кворум

- Строгие И-запросы – это слишком жестко
- На практике нужен т.н. «мягкий AND» (*soft AND*)
- Можно реализовать по-разному, напр. через т.н. *кворум*
- Идея кворума – оставляем только те документы, у которых сумма IDF (т.е. наш *IdfScore*) найденных в документе слов больше какого-то порога *QuorumThreshold*
- Кворум можно реализовывать по-разному, например какой-то модификацией алгоритма пересечения сортированных списков (это нетривиально!)

Запрос «что есть вечером»

Предположим, что известны IDFы:

- $IDF(ЧТО) = 2$
- $IDF(ЕСТЬ) = 5$
- $IDF(ВЕЧЕРОМ) = 10$

и задан $QuorumThreshold = 9$

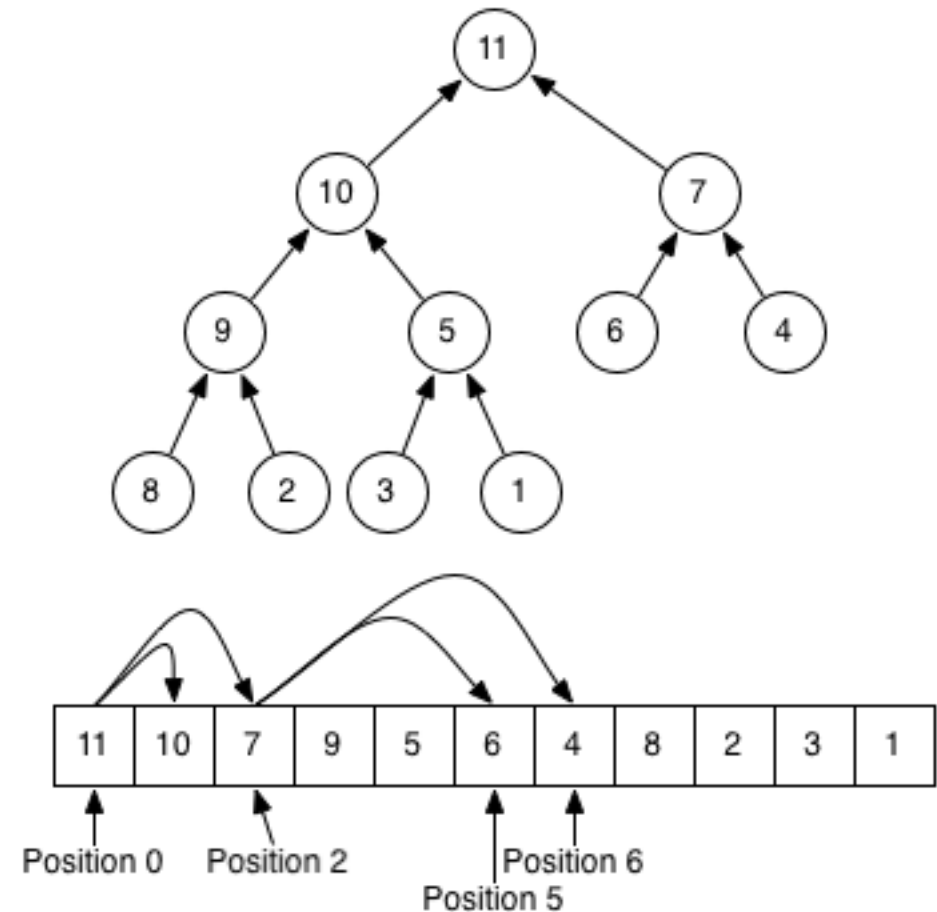
Ищем индексе из 2х документов:

1. Содержит термины ЧТО и ЕСТЬ: $IdfScore(q,d1) = 2 + 5 = 7 < QuorumThreshold$
2. содержит термины ЕСТЬ и ВЕЧЕРОМ: $IdfScore(q,d2) = 5 + 10 = 15 > QuorumThreshold$

Только документ №2 пройдет кворум!

Топ-K максимальных

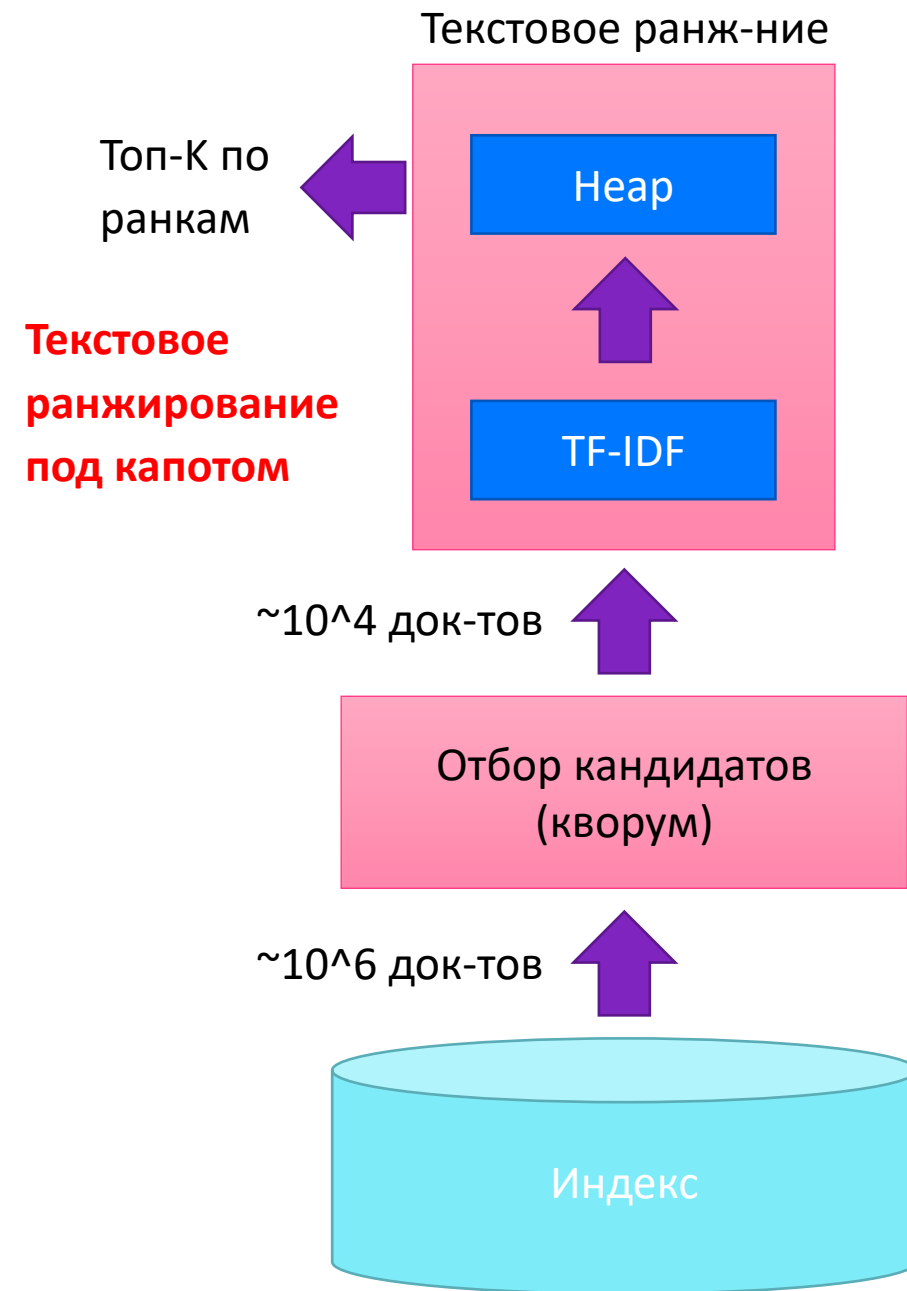
- Допустим, мы отобрали кандидатов и посчитали для них $TfIdfScore(q,d)$
- Теперь надо найти Топ-K документов с максимальными скорями, чтобы показать их пользователю
- На практике обычно $K=10$
- Как это сделать?



Структура данных которая могла бы нам помочь...

Кворум + heap

- Для выбора Топ-К документов идеально подходит структура данных *heap* (она же *куча* или *пирамида*)
- Реализуем с ее помощью очередь с приоритетами размером К, в которой в качестве приоритета используется ранк документа (т.е. *TfidfScore*)
- Добавляем все прошедшие кворум кандидаты в нашу очередь
- В конце достаем из очереди Топ-К документов с самыми большими ранками
- Алгоритмическая сложность решения: $O(N_q * \log K)$, где N_q – это число документов, которые прошли кворум
- Это онлайн-алгоритм! Т.е. не надо хранить все ранки
- Именно такое решение работает на этапе черного ранжирования в поиске VK



Модель векторного пространства



Скалярное произведение

- Еще раз внимательно посмотрим на формулу $TfIdfScore$...
- Пусть V – это словарь всех известных нам терминов, а M – размер этого словаря, т.е. полное число всех известных нам терминов. Отдельные термины из словаря будем обозначать v_i
- Представим запрос и документ в виде векторов \mathbf{q} и \mathbf{d} размером M :
 - $\mathbf{q} = (0, 1, 0, 0, 1, \dots, 0, 1)$ где компонент q_i на i -й позиции равен 1 если i -й термин v_i из словаря присутствует в запросе, и 0 – если нет
 - $\mathbf{d} = (\dots, TF(v_i, d) * IDF(v_i), \dots)$ где компонент d_i на i -й позиции равен TF-IDF i -го термина v_i из словаря (и, очевидно, он 0 если такого термина нет в документе)
- Тогда: $TfIdfScore(q, d) = \mathbf{q} * \mathbf{d}$
- У нас скалярное произведение!

$$TfIdfScore(q, d) = \sum_{t_i \in q} TF(t_i, d) * IDF(t_i)$$

Модель векторного пространства (VSM)

- Идея: представляем запросы и документы в едином векторном пространстве
- Ранжируем документы по скалярному произведению (или косинусу) между векторами запроса и документа
- Очень мощная идея которая обобщает «наивный» TF-IDF

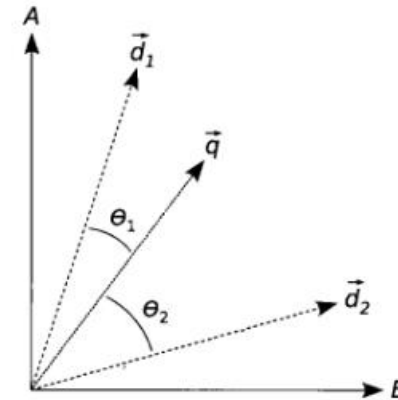


Figure 2.8 Document similarity under the vector space model. Angles are computed between a query vector \vec{q} and two document vectors \vec{d}_1 and \vec{d}_2 . Because $\theta_1 < \theta_2$, d_1 should be ranked higher than d_2 .

Иллюстрация идеи векторного пространства из книги Butcher'а, ищем косинусное расстояние между запросом **q** и документами **d1** и **d2**.

Варианты TF-IDF

- Существует множество способов представить запрос **q** и документ **d** в виде векторов, которые, по сути, являются вариантами TF-IDF
- Например:
 - Вектора **q** и **d** можно нормализовать (в этом случае $\cos(\mathbf{q}, \mathbf{d}) = \mathbf{q} * \mathbf{d}$) – тогда скоры будут инварианты относительно удвоения текстов
 - В векторе **q** вместо 1/0 тоже можно использовать TF слов в запросе (хорошо для очень длинных запросов!)
 - В формуле TF тоже можно использовать логарифм
 - TF и IDF можно сглаживать, причем по-разному
 - ... и многое другое

Частота термина	Документная частота	Нормировка
n (natural) $tf_{t,d}$	n (no) 1	n (none) 1
l (logarithm) $1 + \log(tf_{t,d})$	t (idf) $\log \frac{N}{df_t}$	c (cosine) $\frac{1}{\sqrt{w_1^2 + w_2^2 + \dots + w_M^2}}$
a (augmented) $0,5 + \frac{0,5tf_{t,d}}{\max_t (tf_{t,d})}$	p (prob idf) $\max \left[0, \log \frac{N - df_t}{df_t} \right]$	u (pivoted unigue) $1/u$ (раздел 17.4.4)
b (boolean) $\begin{cases} 1, \text{если } tf_{t,d} > 0, \\ 0 - \text{в противном случае} \end{cases}$		b (byte size) $1/CharLength^a, a < 1$
L (log ave) $\frac{1 + \log(tf_{t,d})}{1 + \log(ave_{red}(tf_{t,d}))}$		

Рис. 6.15. Система обозначений SMART для вариантов *tf-idf*. Здесь *CharLength* — количество символов в документе

Варианты TF-IDF (взято из книги Маннинга)

TF-IDF weighting

- Мы научились использовать векторы TF-IDF для задачи *текстового ранжирования*
- Но векторное представление текстов широко применяется не только в поиске, но и в других задачах NLP (напр. задаче классификации текстов)
- Нам надо как-то подать текст на вход модели, а для этого надо представить его в виде чисел
- Такое преобразование текста называется *векторизацией*
- Существуют множество способы векторизации, но на практике в качестве первого решения чаще всего используют векторы TF-IDF (*)
- Такая векторизация часто называется TF-IDF weighting (или представление текста признаками, взвешенными по TF-IDF)

(*) это в первую очередь касается классического ML, в современном NLP чаще всего используют другие подходы такие как subword tokenization

Examples

```
>>> from sklearn.feature_extraction.text import TfidfVectorizer
>>> corpus = [
...     'This is the first document.',
...     'This document is the second document.',
...     'And this is the third one.',
...     'Is this the first document?',
... ]
>>> vectorizer = TfidfVectorizer()
>>> X = vectorizer.fit_transform(corpus)
>>> vectorizer.get_feature_names_out()
array(['and', 'document', 'first', 'is', 'one', 'second', 'the', 'third',
       'this'], ...)
>>> print(X.shape)
(4, 9)
```

Пример векторизации документов на практике с помощью *TfidfVectorizer* из библиотеки **scikit-learn**

Векторизация: пример

- Посмотрим как можно векторизовать маленькую коллекцию текстов
- У нас $M = 9$ терминов и $N = 7$ документов
- Показаны как матрица *термин-документ* (знакомая нам из лекции про обратный индекс), так и векторные представления документов
- Векторы документов тоже удобно представить в виде столбцов матрицы $M \times N$
- В данном примере автор использовал только TF (т.е. считал что все $IDF = 1$), а векторы документов были нормализованы с использованием L2-нормализации
- Обратите внимание, что даже для такой маленькой коллекции матрицы и векторы уже очень разрежены!

Коллекция
документов
и словарь
терминов

Terms	Documents
T1: Bab(y,ies,y's)	D1: <u>Infant</u> & <u>Toddler</u> First Aid
T2: Child(ren's)	D2: <u>Babies</u> & <u>Children's</u> Room (For Your <u>Home</u>)
T3: Guide	D3: <u>Child</u> <u>Safety</u> at <u>Home</u>
T4: Health	D4: Your <u>Baby's</u> <u>Health</u> and <u>Safety</u> : From <u>Infant</u> to <u>Toddler</u>
T5: Home	D5: <u>Baby</u> <u>Proofing</u> Basics
T6: Infant	D6: Your <u>Guide</u> to Easy Rust <u>Proofing</u>
T7: Proofing	D7: Beanie <u>Babies</u> Collector's <u>Guide</u>
T8: Safety	
T9: Toddler	

The 9×7 term-by-document matrix before normalization, where the element \hat{a}_{ij} is the number of times term i appears in document title j :

Матрица
термин-
документ

$$\hat{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Вектор
документа

The 9×7 term-by-document matrix with unit columns: D7

Векторные
представления

$$A = \begin{pmatrix} 0 & 0.5774 & 0 & 0.4472 & 0.7071 & 0 & 0.7071 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0.7071 \\ 0 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0 \\ 0 & 0 & 0.5774 & 0.4472 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \end{pmatrix}$$

Bag of Words (BoW)

- TF-IDF и другие модели векторного пространства позволяют получить неплохое ранжирование «из коробки», без машинного обучения
- Но у них есть много недостатков
- Например, TF-IDF – это т.н. bag of words модель
- Важно учитывать еще порядок и близость слов
- Как именно это сделать – поговорим в следующей лекции

- Эти документы всегда будут иметь один и тот же *TfidfScore*:

1. МФТИ лучше чем МГУ
2. МГУ лучше чем МФТИ

Но, очевидно, смысл этих документов отличается!

Lexical mismatch

- $TfIdfScore(q,d)$ хорошо «работает» только для *лексического* поиска по ключевым словам т.к. формуле ненулевые слагаемые будут только у терминов, которые есть и в запросе, и в документе
 - Но поиска по ключевым словам часто недостаточно
 - Проблема №1: *синонимия*
 - Проблема №2: *полисемия (омонимия)*
 - Эти (и похожие) проблемы часто называют проблемой *lexical mismatch* (или *vocabulary mismatch*)
 - *Синонимия*: как найти документ про САНТА КЛАУСА по запросу ДЕД МОРОЗ?
 - *Полисемия (омонимия)*: ЯГУАР – это животное, машина или напиток? (*)
- (*) В лингвистике, строго говоря, различают полисемию, омонимию и т.д., но мы в такие тонкости вдаваться не будем ([википедия](#))

Синонимы

- Наивный способ решения проблемы синонимии: *расширение запроса синонимами (query expansion)*
- Классическая задача вычислительной лингвистики
- Неплохо работает в простых случаях (ПОДЪЕЗД == ПАРАДНАЯ)
- Хорошо «ложится» на булев поиск (ИЛИ-запросы)
- Проблема: не масштабируемо
- Хочется как-то понимать, что тексты похожи по смыслу
- Нам нужна т.н. *семантическая близость (semantic similarity)*
- Что решить эту задачу, нам придется погрузиться в семантику

Пример: запрос «*московский международный конкурс юных музыкантов*» превращается в:

(МОСКОВСКИЙ | МОСКВА)
(МЕЖДУНАРОДНЫЙ | ВСЕМИРНЫЙ | ИНТЕРНАЦИОНАЛЬНЫЙ | МИРОВОЙ)
(КОНКУРС | КОНКУРСНЫЙ | ОЛИМПИАДА | СОРЕВНОВАНИЕ | СОСТЯЗАНИЕ)
(ЮНЫЙ | МОЛОДЕНЬКИЙ | МОЛОДОЙ | ЮВЕНИЛЬНЫЙ | ЮНОСТЬ)
(МУЗЫКАНТ | МУЗЫКАНТОВ | АРТИСТ | БАРАБАНЩИК | КОМПОЗИТОР | МУЗЫКАНТСКИЙ | ПИАНИСТ | САКСОФОНИСТ)

Очевидные проблемы:

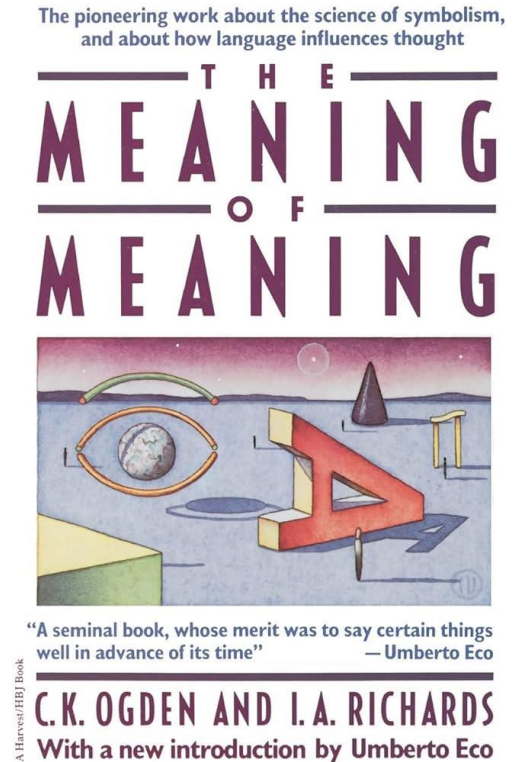
- когда остановиться: МУЗЫКАНТ и ПИАНИСТ – это все еще синонимы, или уже нет?
- веса синонимов: ЮНЫЙ и МОЛОДОЙ наверное «ближе» по смыслу, чем ЮНЫЙ и ЮВЕНИЛЬНЫЙ

Семантика



Семантика?

- Нам нужна семантика, или смысл
- Но что такое «смысл»?
- Где-то здесь начинается философия
- Будем максимально практичны (и всегда помнить про ДЕДА МОРОЗА!)



Пытаться дать определения понятиям семантика, смысл, значение и т.п. может стать full-time job если вы задумываетесь о карьере профессионального философа

(на картинке обложка книги ***The Meaning of Meaning*** by C. K. Ogden and I. A. Richards, см. https://en.wikipedia.org/wiki/The_Meaning_of_Meaning)

Семантическая близость

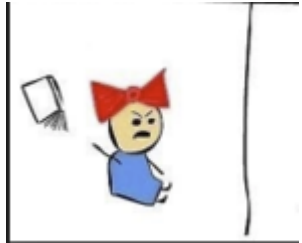
- Задача: выразить семантику вычислительно
- Близкие по смыслу тексты q и d должны получить близкие скоры семантической близости $SemScore(q,d)$
- Например: $SemScore(ДЕД МОРОЗ, САНТА КЛАУС) \gg SemScore(ДЕД МОРОЗ, БАБА ЯГА)$
- Но как быть с:
 - $SemScore(ДЕД МОРОЗ, СНЕГУРОЧКА)?$
 - $SemScore(ДЕД МОРОЗ, ЗИМА)?$
 - $SemScore(ЗИМА, ВЕСНА)?$



Дед Мороз и Баба Яга (!) глазами Kandinsky 3.1

Семантика – это сложно!

- Семантическая близость может быть разной, например:
 - синонимы (ДЕД МОРОЗ и САНТА КЛАУС)
 - тематическая близость (ДЕД МОРОЗ и ЗИМА)
 - категорийная близость (ЗИМА и ВЕСНА – это ВРЕМЕНА ГОДА)
 - и т.д.
- В разных задачах могут быть важны разные аспекты семантики
- Пока даже непонятно как к этому подойти
- Дальше пойдем по пути развития интуиции



Дистрибутивная гипотеза

- Будем опираться на дистрибутивную гипотезу: *слова и выражения, встречающиеся в схожих контекстах, близки по смыслу*
- Пример:
 - Дед Мороз – это символ Нового Года
 - Санта Клаус – это символ Нового Года
- => «ДЕД МОРОЗ» и «САНТА КЛАУС» близки по смыслу
- Была предложена Джоном Фертом (John Firth) в 50-е годы
- Лежит в основе современного NLP

Скрытая реальность

- За каждым словом обычно стоит какая-то абстрактная «идея» или «концепция» (concept)
- Идеи можно выражать словами по-разному
- Например идею «деда мороза» можно выразить как: «дед мороз» или «санта клаус» или «волшебный персонаж который приносит детям на новый год подарки» или «йоулупукки» (финский дед мороз) и т.д.
- У нас есть 2 реальности: **скрытая** реальность абстрактных идей и **видимая** реальность слов, выражающих эти идеи
- Скрытая реальность *порождает* видимую
- Выбор же конкретных слов при этом во многом случаен!

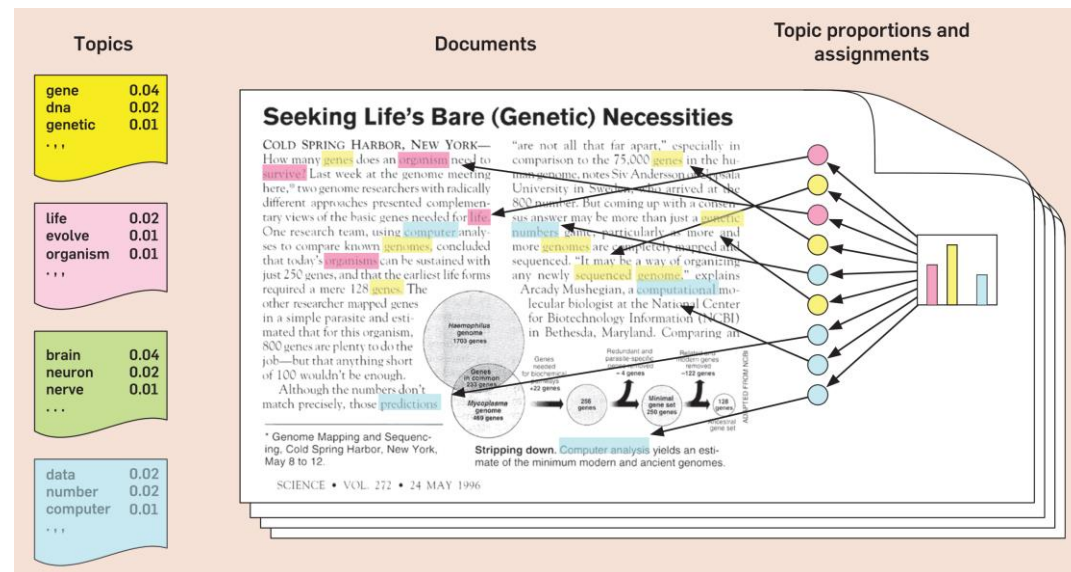


Абстрактный Дед Мороз в вакууме
глазами Kandinsky 3.1

Абстрактные идеи можно выражать не только словами, но и картинками.

Документы и тематики

- Перейдем от отдельных слов к документам (текстам)
- Как правило, каждый осмысленный документ можно отнести к каким-то *тематикам* или *темам* (на английском *topics*)
- Верно и обратное: разные документы можно отнести к одной и той же тематике
- Выбор слов в каждом конкретном документе во многом случаен – все те же идеи можно бы было выразить и другими словами!
- С другой стороны, факт наличия того или иного слова является «сигналом» того, что документ относится к определенной тематике



Документ, который можно отнести к нескольким тематикам: *генетика, эволюция, нервная система, компьютеры, ...*

Слова (такие как *genetic*) являются маркерами принадлежности к тематике.

Картинка с [сайта pyro.ai](http://сайта.pyro.ai)

Тематики описываются словами

- Интуитивно, каждому слову можно назначить некий вес, которые определяет насколько хорошо данная тематика описывается данным словом
- Типичный пример – *облака слов (word clouds)*
- Чем важнее слово – тем больше и жирнее шрифт на картинке



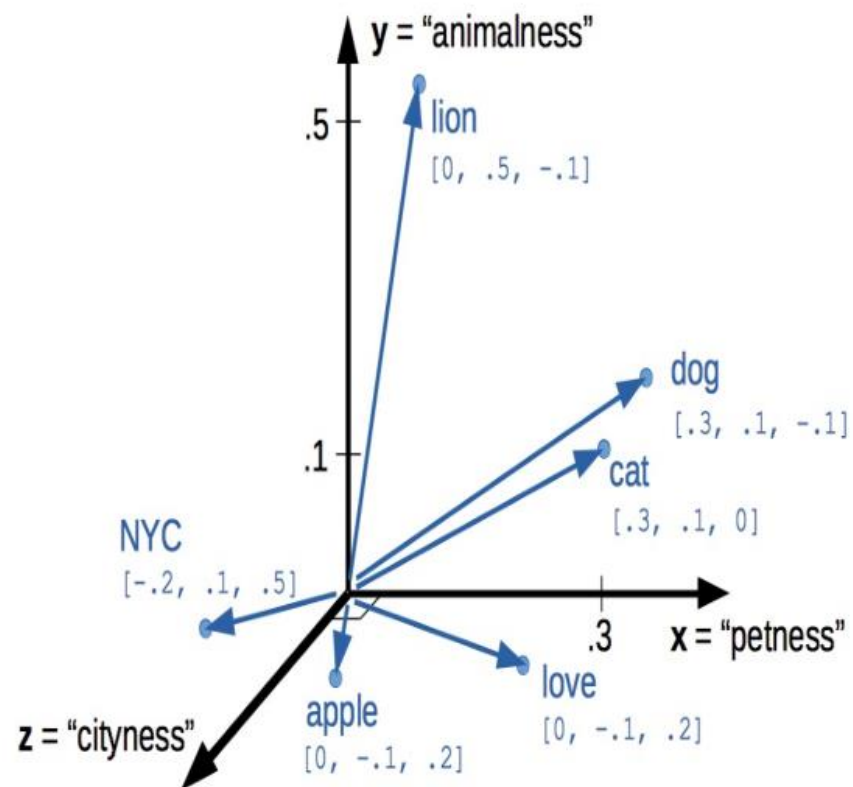
Тематика «Утро»



Тематика «Образование»

Слова относятся к тематикам

- Верно и обратное: каждому слову хочется назначить некую меру сродства к каждой из тематик
- Таким образом, мы имеем связь в обе стороны: слово \Leftrightarrow тематика
- Это становится похожим на нашу *матрицу термин-документ*!

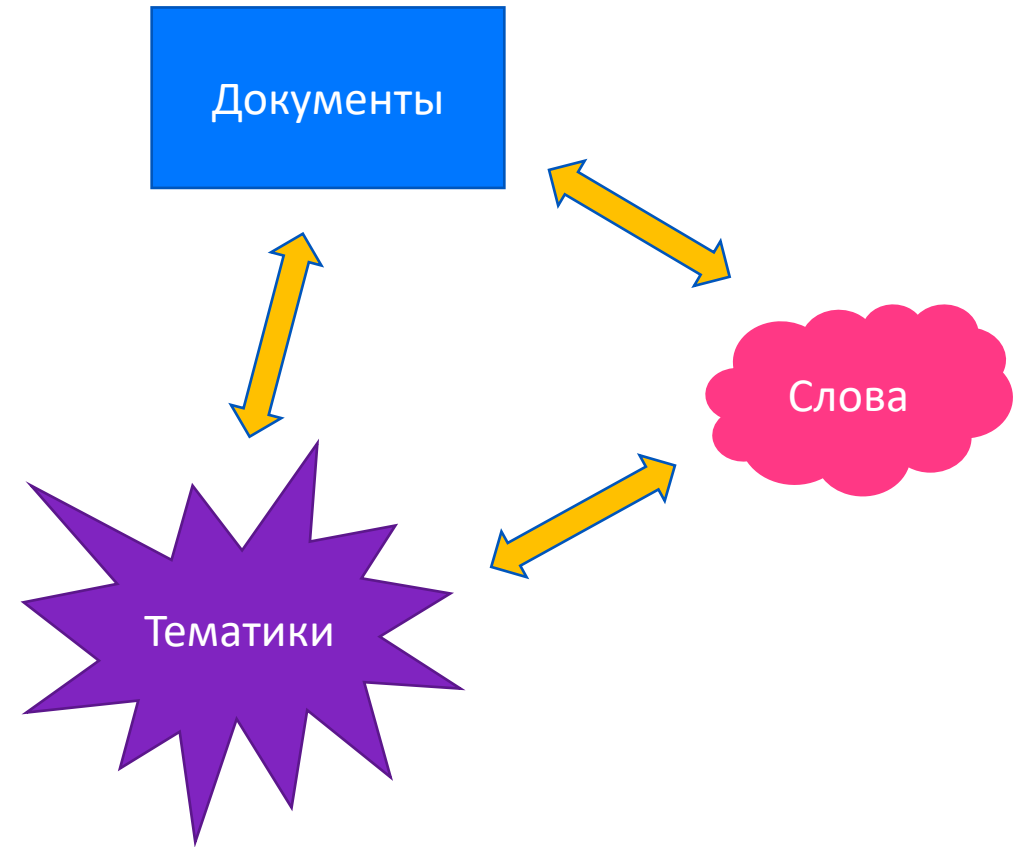


Сродство слов (таких как *LION*) к тематикам (таким как *ANIMALNESS*)

Картинка из книги [NLPiA](#)

Слова \Leftrightarrow Документы \Leftrightarrow Тематики

- Соберем все вместе
- Связь слов, документов и тематик хочется выразить в виде 3-х матриц:
 - Слово (термин) \Leftrightarrow документ
 - Документ \Leftrightarrow тематика
 - Тематика \Leftrightarrow слово
- Матрицы содержат веса – степени сродства (и пока не важно откуда эти веса берутся)
- Только матрица *слово \Leftrightarrow документ* задана в явном виде, а две другие скрыты!
- Чтобы сделать скрытое явным, надо научиться как-то выделять тематики



Матрица термин-документ с точки зрения ML

- Посмотрим на матрицу термин-документ глазами дата-саентиста
- Видим объекты и признаки (фичи)
- Наша матрица размером $M \times N$, где M – это число всех известных терминов (размер словаря), а N – число документов
- Каждый термин (слово) описан признаками: фактами принадлежности к N -му документу
- Но и каждый документ описан признаками: фактами наличия в документе M -го слова!

Коллекция документов и словарь терминов

Terms	Documents
T1: Bab(y,ies,y's)	D1: <u>Infant</u> & <u>Toddler</u> First Aid
T2: Child(ren's)	D2: <u>Babies</u> & <u>Children's</u> Room (For Your <u>Home</u>)
T3: Guide	D3: <u>Child</u> <u>Safety</u> at <u>Home</u>
T4: Health	D4: Your <u>Baby's</u> <u>Health</u> and <u>Safety</u> : From <u>Infant</u> to <u>Toddler</u>
T5: Home	D5: <u>Baby</u> <u>Proofing</u> Basics
T6: Infant	D6: Your <u>Guide</u> to Easy Rust <u>Proofing</u>
T7: Proofing	D7: Beanie <u>Babies</u> Collector's <u>Guide</u>
T8: Safety	
T9: Toddler	

The 9×7 term-by-document matrix before normalization, where the element \hat{a}_{ij} is the number of times term i appears in document title j :

Матрица термин-документ

$$\hat{A} = \begin{pmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

The 9×7 term-by-document matrix with unit columns:

Фичи!

Матрица термин-документ (после L2-норм.)

$$A = \begin{pmatrix} 0 & 0.5774 & 0 & 0.4472 & 0.7071 & 0 & 0.7071 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0.7071 \\ 0 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0 \\ 0 & 0 & 0.5774 & 0.4472 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \end{pmatrix}$$

Пример матрицы термин-документ из книги *Berry & Browne* (см. список литературы)

Cosine Similarity

- Согласно модели векторного пространства, косинус между векторами документов выражает их близость:

$$TfIdfScore(\mathbf{d}_i, \mathbf{d}_j) = \cos(\mathbf{d}_i, \mathbf{d}_j) = \frac{\mathbf{d}_i * \mathbf{d}_j}{\|\mathbf{d}_i\| * \|\mathbf{d}_j\|}$$

- (для нормализованных векторов косинус равен скалярному произведению)
- Эта близость лексическая, т.к. в векторах \mathbf{d} не-нулевые веса имеют только те элементы, которые соответствуют словам данного документа
- Аналогично, косинус между векторами терминов $\cos(\mathbf{t}_i, \mathbf{t}_j)$ можно понимать как меру близости слов!

$$A = \begin{pmatrix} 0 & 0.5774 & 0 & 0.4472 & 0.7071 & 0 & 0.7071 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 & 0.4472 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0.4472 \\ 0 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 & 0.4472 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0 & 0.4472 \\ 0 & 0 & 0.5774 & 0.4472 & 0 & 0 & 0 & 0.4472 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Векторы \mathbf{d}_1 и \mathbf{d}_2 близки т.к. имеют 2 общих слова: $\cos(\mathbf{d}_1, \mathbf{d}_2) \approx 0.67$

Разреженность

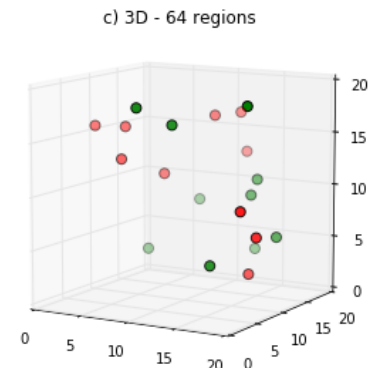
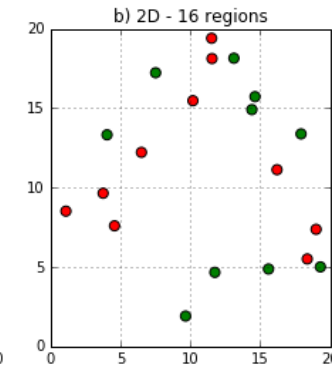
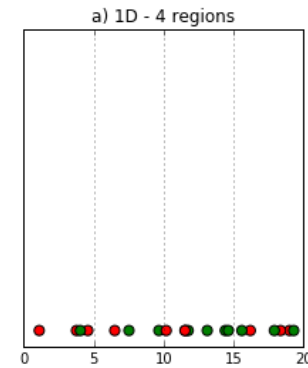
- Наши матрицы и векторы очень разрежены (sparse)
- Типичные размеры матрицы термин-документ:
 - Размер словаря: $M \approx 10^5 - 10^6$
 - Размер корпуса текстов: $N > 10^6$ (и до бесконечности)
- Типичное ненулевых элементов в документе: $D_{nz} \approx 10 - 10^3$

$$A = \begin{pmatrix} 0 & 0.5774 & 0 & 0.4472 & 0.7071 & 0 & 0.7071 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 & 0.4472 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0.4472 \\ 0 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 & 0.4472 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0 & 0.4472 \\ 0 & 0 & 0.5774 & 0.4472 & 0 & 0 & 0 & 0.4472 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 & 0 \end{pmatrix}$$

Даже на примере маленькой коллекции 7x9 видно, что нули доминируют

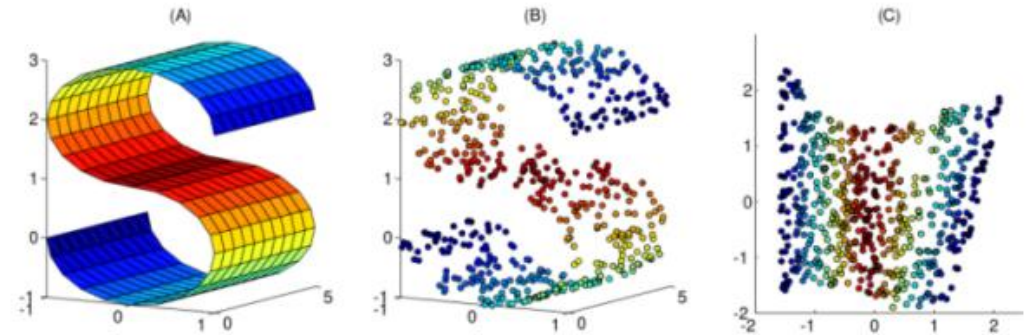
Многомерность

- Размерность наших векторов огромна!
- Чем плоха такая ситуация? (подсказка на картинке)



Проклятие размерности

- Проблема «проклятия размерности» – чем больше размерность ваших данных, тем сложнее с ними работать
- Возникает желание как-то уменьшить размерность
- Такая задача называется задачей *снижения размерности* (*dimensionality reduction*) ([википедия](#))
- Снижение размерности позволяет проявить скрытые закономерности, которые были «размыты» в исходном пространстве высокой размерности
- Какие способы снижения размерности вы помните/знаете?




Снижение размерности на примере т.н. S-кривой (S-curve)

Картинка из блога [wildart](#)

Документы как векторы тематик

- Посмотрим документы как на векторы из K тематик
- Обычно $K \ll M$, где M – размер словаря
- Косинус между такими векторами будет выражать близость в пространстве тематик что можно понимать как *семантическую близость*
- Выбор слов случаен, увеличивает размерность и «размывает» смысл
- Документы, близкие по смыслу (т.е. близкие в «пространстве тематик») могут сильно разойтись в «пространстве слов»
- И наоборот – представление документов в виде векторов из тематик «проявляет» этот скрытый смысл
- Подозрительно похоже на уменьшение размерности, не так ли?



Document–topic Matrix (DT)				
	Topic 1	Topic 2	.	Topic k
Document 1	0.72	0.24	.	0.08
Document 2	0.04	0.60	.	0.39
Document 3	0.15	0.09	.	0.11
Document 4	0.51	0.70	.	0.24
.
Document d	0.61	0.23	.	0.28

Матрица документ-тематика размером $N \times K$, где K – это число тематик

Идея

- Будем уменьшать размерность матрицы термин-документ!
- Но не просто так, а хитро
- Мы должны получить представления документов в виде плотных (dense) векторов
- Отдельные компоненты этих векторов должны соответствовать тематикам
- Косинусное расстояние между такими векторами все еще должно отражать близость, но уже не только лексическую, но и семантическую
- На этом с философией мы заканчиваем, и переходим к математике!

Матричные разложения



Собственные векторы

- Придется вспомнить немного линейной алгебры
- *Собственный вектор* (квадратной) матрицы A – это ненулевой вектор, умножение которого на матрицу A даёт коллинеарный вектор: $Ax = \lambda x$
- Число λ называется *собственным значением*; говорят что вектор x соответствует собственному значению λ

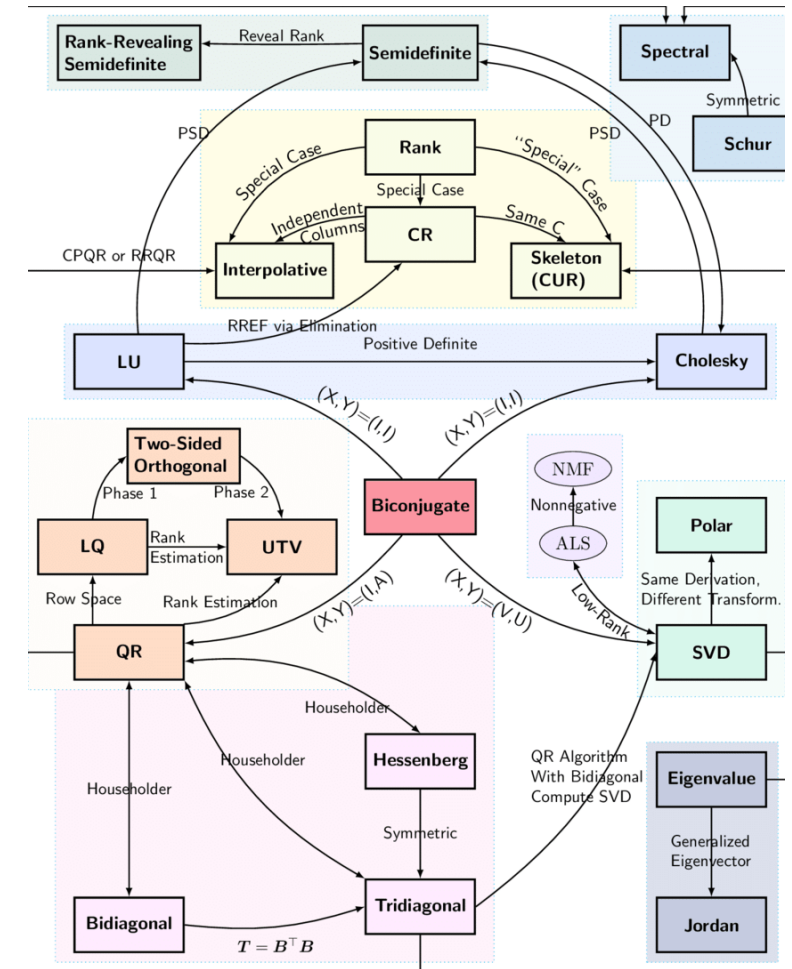
$$\begin{array}{ccc} \begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix} \begin{bmatrix} 1/2 \\ 1/2 \\ 1 \end{bmatrix} & = & 4 \begin{bmatrix} 1/2 \\ 1/2 \\ 1 \end{bmatrix} \\ A & \text{eigenvalue} & \text{eigenvector} \end{array} \quad \begin{array}{ccc} \begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} & = & -2 \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \\ \begin{bmatrix} 1 & -3 & 3 \\ 3 & -5 & 3 \\ 6 & -6 & 4 \end{bmatrix} \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} & = & -2 \begin{bmatrix} -1 \\ 0 \\ 1 \end{bmatrix} \end{array}$$

Пример матрицы A , ее собственных значений и векторов.

Обратите внимание что в данном случае собственному значению $\lambda = 2$ соответствуют два независимых собственных вектора.

Матричные разложения

- Разложение матрицы – представление матрицы A в виде произведения матриц, обладающих определенными свойствами, такими как, например, *ортогональность*, *симметричность* или *диагональность*
- На английском матричные разложения называются *matrix decompositions* или *matrix factorizations*
- Существуют самые разные, в зависимости от области применения
- Нас будет интересовать в первую очередь *сингулярное разложение* (*Singular Value Decomposition* или просто *SVD*)



«Карта» матричных разложений (взято из [статьи](#))

Сингулярное разложение (SVD)

- Определение: $X = U\Sigma V^t$, где:
 - Матрица X – это произвольная матрица размера $M \times N$
 - Матрица U – это ортогональная матрица размера $M \times M$. Столбцы матрицы U это т.н. левые *сингулярные векторы*, т.е. собственные векторы матрицы XX^t
 - Матрица V^t – это ортогональная размера $N \times N$. Столбцы матрицы U это т.н. правые *сингулярные векторы*, т.е. собственные векторы матрицы X^tX
 - Матрица Σ имеет размер $M \times N$. Все ее элементы, лежащие на главной диагонали – это т.н. *сингулярные значения (или числа)*, а все элементы, не лежащие на диагонали – нулевые
 - Сингулярные значения $\Sigma_{ii} = \sigma_i = \sqrt{\lambda_i}$, где:
 - i лежит в диапазоне $1 \leq i \leq r$, где r – это ранг матрицы X
 - $\lambda_i \geq \lambda_{i+1}$ (т.е. сингулярные значения упорядочены по убыванию)
 - λ_i – это собственные значения матриц XX^t и X^tX

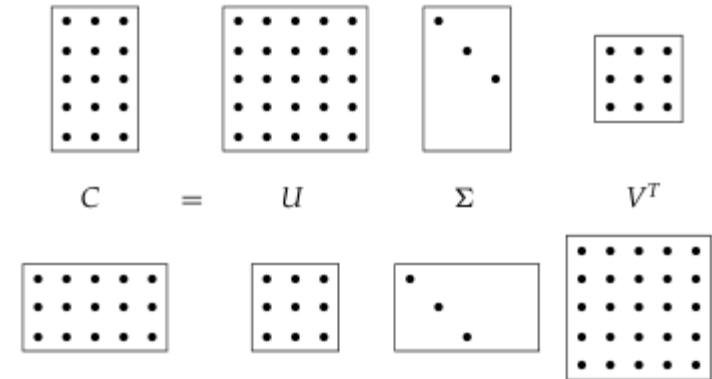


Иллюстрация SVD разложения матрицы C размера $M \times N$.

Рассмотрены два случая: $M > N$ и $M < N$

(картинка из книги Маннинга)

SVD разложение: пример

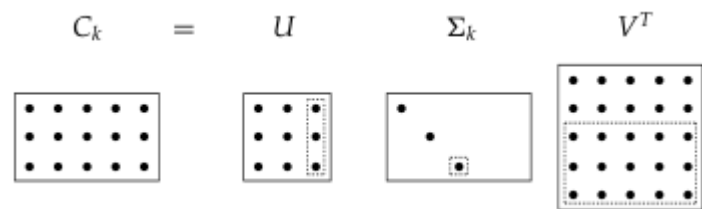
- Раскладываем матрицу размера 4x2 как произведение 3-х матриц: 4x2, 2x2 и 2x2.
- Сингулярные значения $\sigma_1 = 2.236$ и $\sigma_2 = 1.000$
- На практике можно воспользоваться, например, реализацией из библиотек *numpy* (`numpy.linalg.svd`) или *scipy* (`scipy.linalg.svd`)

$$C = \begin{pmatrix} 1 & -1 \\ 0 & 1 \\ 1 & 0 \\ -1 & 1 \end{pmatrix} = \begin{pmatrix} -0.632 & 0.000 \\ 0.316 & -0.707 \\ -0.316 & -0.707 \\ 0.632 & 0.000 \end{pmatrix} \begin{pmatrix} 2.236 & 0.000 \\ 0.000 & 1.000 \end{pmatrix} \begin{pmatrix} -0.707 & 0.707 \\ -0.707 & -0.707 \end{pmatrix}$$

Пример из книги Маннинга.

Приближение с помощью SVD-разложения

- С помощью SVD-разложения для любой матрицы X можно получить ее приближение
- Процедура выглядит так:
 - Применяем SVD-разложение: $X = U\Sigma V^t$
 - Получаем из матрицы Σ новую матрицу Σ_k путем зануления $r-k$ наименьших сингулярных значений
 - Вычисляем матрицу $X_k = U\Sigma_k V^t$ которая и будет нашим приближением
- Можно показать, что такое приближение является в некотором смысле оптимальным

$$C_k = U \Sigma_k V^T$$


Приближение матрицы C матрицей C_k .

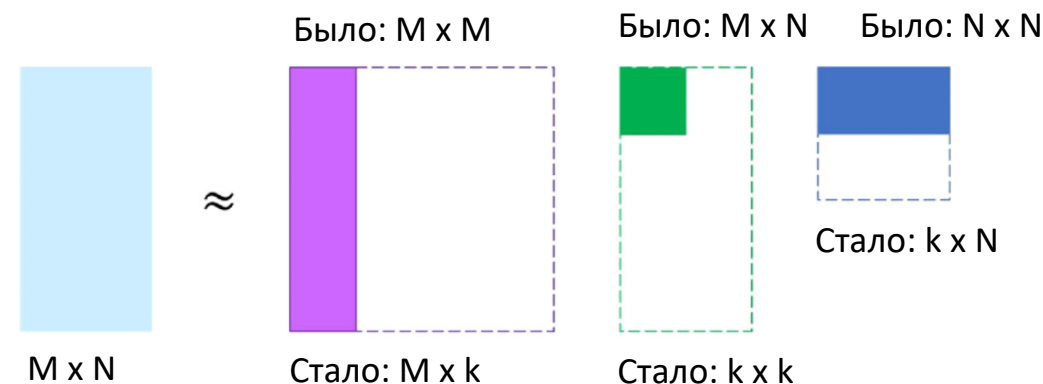
Получается занулением 3-го (наименьшего) сингулярного значения.

Обратите внимание, что в таком случае последний столбец матрицы U и последние 3 строки матрицы V^t становятся «лишними» и их можно отбросить без потери точности.

(картинка из книги Маннинга)

Truncated SVD

- Последний пример приводит нас к идее сокращенного (или компактного) представления – это т.н. *truncated SVD* (иногда его еще называют *reduced SVD*)
- Заменяем нашу Σ_k диагональной матрицей размером $k \times k$, где k – это число ненулевых сингулярных значений
- Отбрасываем $(M - k)$ правых столбцов U и, аналогично, $(N - k)$ последних строк V^t
- И дальше работаем только с сокращенными матрицами Σ_k , U и V^t
- Такое представление экономит нам много памяти в типичном случае когда $k \ll r < \min(M, N)$



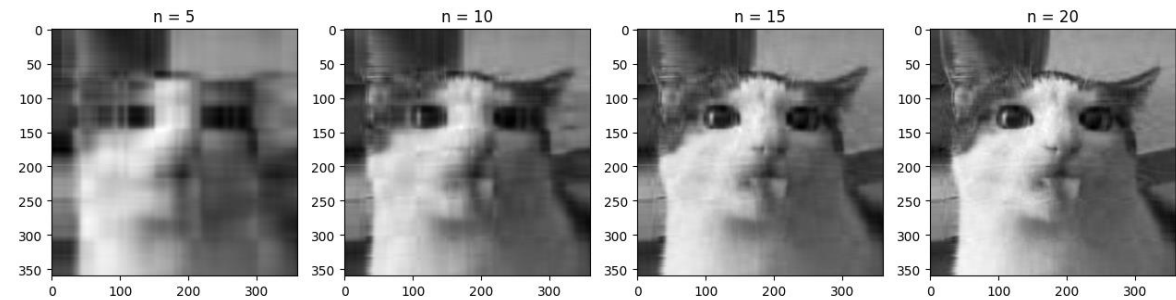
Truncated SVD позволяет драматически уменьшить размеры матриц.

Оптимальность SVD

- Определим фробениусову норму матрицы X как:

$$\|X\|_F = \sqrt{\sum_{i=1}^M \sum_{j=1}^N X_{ij}^2}$$

- Предположим, что нам дана матрица X размера $M \times N$ и мы хотим найти матрицу X_k ранга не выше k такую, что фробениусова норма разности $X - X_k$ будет минимальна
- Можно показать, что наилучшая в таком смысле матрица получается в результате SVD-разложения (т.н. теорема Эккарта-Янга, 1936)
- Таким образом, SVD-разложение позволяет получить оптимальное приближение матрицей меньшего ранга (*low-rank approximation*)



Низкоранговые приближения применяются в огромном числе самых разных задач, один из классических примеров – сжатие изображений (с потерей качества).

По сути, таким способом мы можем сжать любую матрицу.

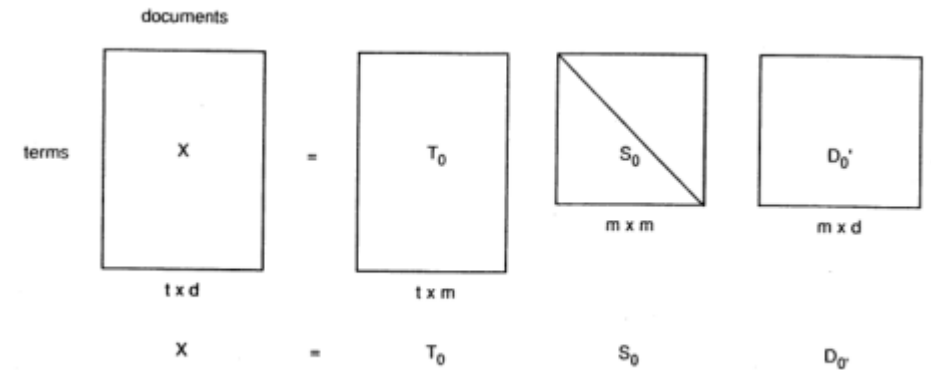
(картинка взята из [поста](#))

Латентно- семантический анализ (LSA)



Разложение матрицы термин-документ

- Применить SVD-разложение для нашей матрицы термин-документ X размера $t \times d$, где t – это общее число терминов (размер словаря), а d – это число документов в корпусе
- В обозначениях как на картинке SVD выглядит: $X = T_0 S_0 D_0^t$
- Каждому термину соответствует строка в матрице T_0 , а каждому документу столбец в матрице D_0^t
- Выглядит красиво, но пока ничего нового мы не получили, это просто другой способ представить матрицу X

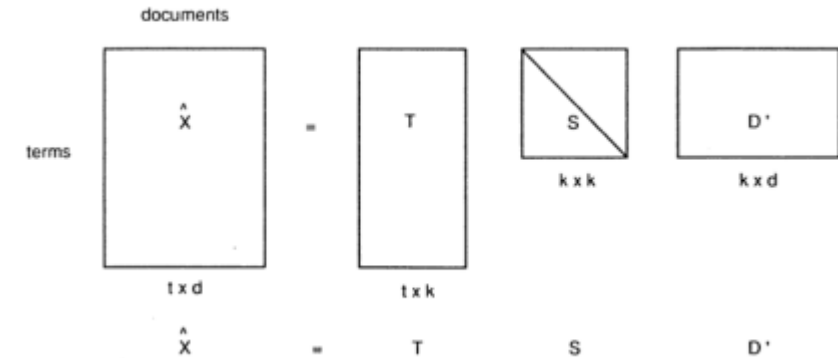


Обратите внимание, что на картинке SVD изображен уже в truncated-виде т.к. матрица S_0 уже квадратная размера $m \times m$ (а не $t \times d$ как в полной форме)

(картинка из статьи *Deerwester, 1990*)

Снизим размерность

- А теперь ключевой момент!
- Применим низкоранговое приближение, оставив только k самых больших сингулярных значений из матрицы S_0 (причем берем $k \ll \min(t, d)$)
- Получаем эффект снижения размерности
- Термины теперь представлены векторами длиной k
- Документы теперь тоже представлены векторами длиной k !
- Обратите внимание, что вектора терминов и документов больше не будут разреженными – по сути, мы получили плотные (dense) эмбединги терминов и документов!
- Восстановленная матрица \hat{X} также больше не будет разреженной



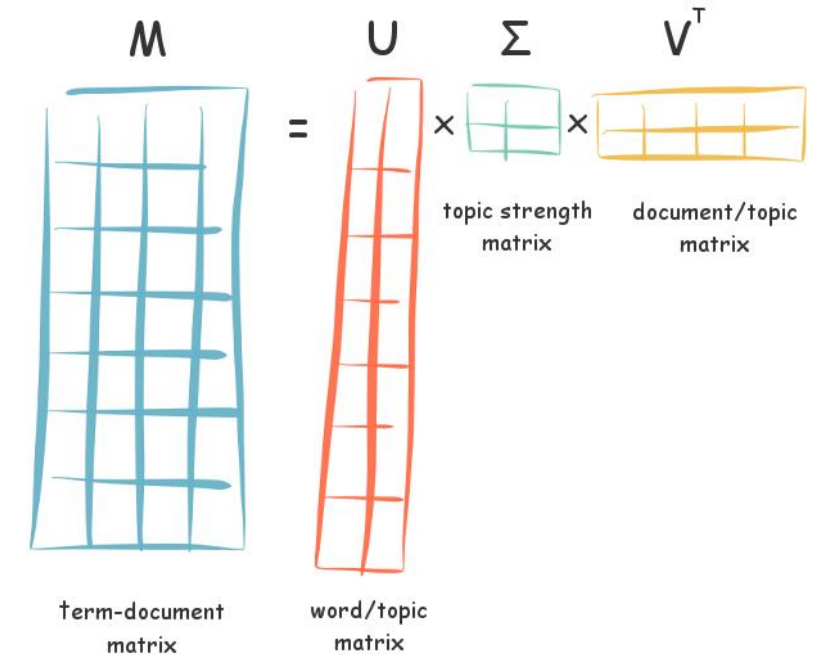
Низкоранговое приближение \hat{X} исходной матрицы X (изображено тоже в truncated-виде).

Т.е. $\hat{X} = TSD^t$, где T , S и D^t это уже «короткие» матрицы (в отличие от T_0 , S_0 и D_0^t на предыдущем слайде)

(картинка из статьи *Deerwester, 1990*)

LSA

- В итоге для терминов у нас:
 - было: sparse-векторы размера d (числа документов)
 - стало: dense-векторы размера k , $k \ll d$
- А для документов:
 - было: sparse-векторы размера t (размера корпуса)
 - стало: dense-векторы размера k , $k \ll t$
- По сути, мы снизили размерность путем «вложения» (эмбединга) наших векторов в некое «скрытое» пространство меньшей размерности
- Магия: элементы векторов в новом пространстве можно понимать как меру сродства терминов и документов к тематикам!
- А сингулярные значения можно понимать как важность данной тематики
- Теперь скалярное произведение таких эмбедингов будет отражать семантическую близость
- Такая техника носит название латентно-семантический анализ (Latent Semantic Analysis, или просто LSA)



SVD разложение позволяет нам «проявить» скрытые тематики и представить матрицу *термин-документ* как произведение матриц *термин-тематика* и *документ-тематика*

(картинка из [поста](#))

LSA: пример, начало

- Рассмотрим метод LSA на примере матрицы термин-документ на картинке справа
- Применим к этой матрице SVD разложение и получим 3 матрицы T_0 , S_0 и D_0^t (не будем их выписывать полностью чтобы не загромождать)
- В частности, матрица S_0 выглядит:

$$\begin{pmatrix} 1.5777 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.2664 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1.1890 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.7962 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5664 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0.1968 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\begin{pmatrix} 0 & 0.5774 & 0 & 0.4472 & 0.7071 & 0 & 0.7071 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.7071 & 0.7071 \\ 0 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0.5774 & 0.5774 & 0 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.7071 & 0.7071 & 0 \\ 0 & 0 & 0.5774 & 0.4472 & 0 & 0 & 0 \\ 0.7071 & 0 & 0 & 0.4472 & 0 & 0 & 0 \end{pmatrix}$$

Матрица X термин-документ размера 9x7

(пример из книги Берри)

Оставим только первые 4 сингулярные значения

LSA: пример, конец

- Наша восстановленная матрица $\hat{X} = TSD^t$ изображена на картинке справа
- Обратите внимание, что она больше не является разреженной!
- Там, где были нули, теперь «проявились» ненулевые значения, которые можно понимать степень сродства данного термина к данному документу
- Т.е., проведя сжатие, мы с одной стороны потеряли часть информации, а с другой – мы нашли скрытые тематики и, с их помощью, как бы «убрали шум», вызванный тем, что исходный набор слов в данном конкретном документе был во многом случаен
- Снижение размерности позволило нам проявить скрытые закономерности, и, потеряв часть информации, открыть новое знание

-0.0018	0.5958	-0.0148	0.4523	0.6974	0.0102	0.6974
-0.0723	0.4938	0.6254	0.0743	0.0121	-0.0133	0.0121
0.0002	-0.0067	0.0052	-0.0013	0.3569	0.7036	0.3569
0.1968	0.0512	0.0064	0.2179	0.0532	-0.0540	0.0532
-0.0723	0.4938	0.6254	0.0743	0.0121	-0.0133	0.0121
0.6315	-0.0598	0.0288	0.5291	-0.0008	0.0002	-0.0008
0.0002	-0.0067	0.0052	-0.0013	0.3569	0.7036	0.3569
0.2151	0.2483	0.4347	0.2262	-0.0359	0.0394	-0.0359
0.6315	-0.0598	0.0288	0.5291	-0.0008	0.0002	-0.0008

Восстановленная матрица \hat{X} термин-документ

(пример из книги Берри)

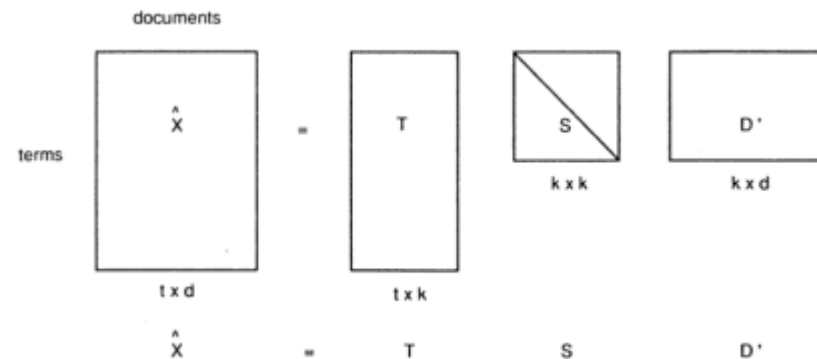
Сравниваем документы

- Теперь у нас есть эмбединги документов, полученные методом LSA
- Как теперь получить скор семантической близости для двух документов?
- Наивный и часто рекомендуемый, но не совсем верный подход: просто берем косинус между эмбедингами документов из матрицы D^t
- Рассмотрим матрицу $X^t X$ – ее элементы содержат скалярные произведения между разреженными документными векторами
- Разумно потребовать, что матрица $\hat{X}^t \hat{X}$ должна приближать эти скалярные произведения
- Простая арифметика дает $\hat{X}^t \hat{X} = DS^2 D^t$ поэтому в качестве представления документа в новом пространстве более правильно взять строки матрицы DS
- Аналогичные формулы можно получить и для сравнения терминов между собой, а также терминов с документами

Нетривиальный момент, хорошо объяснен в статье (Deerwester, 1990)

Эмбеддим новые документы в скрытое пространство

- Самый важный вопрос: как, имея разреженный TF-IDF вектор X_q документа q , заэмбеддить его в наше скрытое пространство?
- Посмотрим внимательно на формулу: $\hat{X} = TSD^t$
- В столбцах матрицы \hat{X} лежат наши («восстановленные») векторы TF-IDF, а в столбцах матрицы D^t – эмбединги документов
- По сути, нам надо «перевернуть» формулу и выразить столбцы D^t через столбцы \hat{X}
- Простая арифметика дает: $D_q^t = X_q^t T S^{-1}$
- Т.е. имея заранее посчитанную матрицу эмбедингов терминов T , для любого TF-IDF вектора X_q легко получить соответствующий ему эмбединг D_q^t (вектор-столбец матрицы D^t)



Latent Semantic Indexing (LSI)

- Как применить все это к поиску?
- Все та же идея векторного пространства: представляем запрос в одном пространстве с документами
- Только вместо разреженных TF-IDF-векторов используем dense эмбединги LSA!
- *LSA* в применении к поиску часто называется латентно-семантическим индексированием (*LSI*)

LSI: полный рецепт

На этапе индексации

- Собираем матрицу термин-документ X (мы это и так всегда делаем при построении обратного индекса)
- Применяем truncated SVD-разложение (со снижением размерности): $X = TSD^t$
- Запоминаем матрицы T и S

На этапе обработки запроса

- Представляем запрос q в виде разреженного TF-IDF вектора X_q
- Эмбеддим запрос в скрытое пространство: $D_q^t = X_q^t T S^{-1}$
- Сравниваем эмбединги запроса и других документов с помощью косинусного расстояния т.е.:
- $\text{SemScore}(q,d) = \cos(D_q^t, D_d^t) (*)$
- Ищем документы, ближайšie к запросу – это и будет результатом поиска!

(*) После масштабирования с использованием матрицы S (см. слайд 70) но на практике этим часто пренебрегают

Что нам дает LSI?

- Предположим, что мы построили нашу поисковую систему на базе LSI
- Мы потеряли часть информации из-за приближения
- На практике, это приводит к тому, что лексический поиск по ключевым словам начинает работать хуже
- Зато мы приобрели семантику!
- Практика показывает, что векторы размером 100-500 элементов уже начинают хорошо отражать семантическую близость
- В частности, они начинают понимать синонимы («ДЕД МОРОЗ» == «САНТА КЛАУС»)
- Лучше всего работает «гибридный» вариант, где документы ранжируются по линейной комбинации лексической и семантической близости, т.е. $\alpha * TfidfScore(q,d) + (1 - \alpha) * SemScore(q,d)$

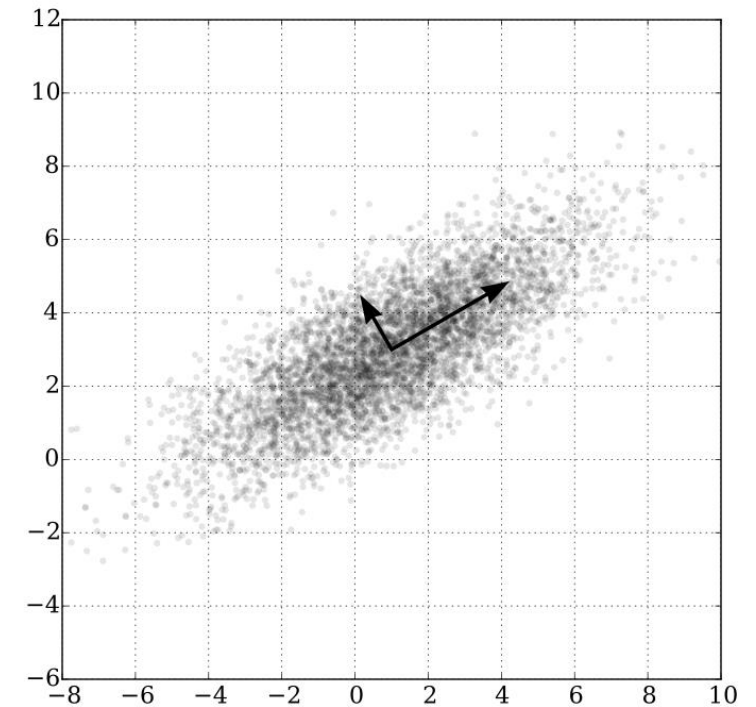
Проблемы LSI

- У метода LSI есть еще одна большая проблема: разлагать большие матрицы долго
- Несмотря на то, что сейчас существуют эффективных алгоритмы разложения больших разреженных матриц, для больших (> 1 млрд. документов) коллекций они неприменимы
- Поэтому на практике используются более современные способы выделения семантики: они и работают быстрее, и результаты получаются гораздо лучше
- Тем не менее, в основе современных подходов лежат все те же проверенные временем идеи (про это еще поговорим)

РСА

- Все, что мы сейчас проделали – это, по сути, метод главных компонент (*PCA, Principal Component Analysis*) в применении к матрице термин-документ (*)
- Классический алгоритм снижения размерности, открытый еще Карлом Пирсоном в 1901 г.
- Почитать про него можно, например, в википедии:
https://en.wikipedia.org/wiki/Principal_component_analysis

(*) за небольшими отличиями в которые сейчас вдаваться не будем



Метод РСА можно понимать как поиск такого ортогонального преобразования в новую систему координат, для которого были бы верны следующие условия:

- Выборочная дисперсия данных вдоль первой координаты максимальна
- Выборочная дисперсия данных вдоль второй координаты максимальна при условии ортогональности первой координате
- и т.д.

Картинка из [википедии](#)

Рекомендательные системы

- Этот же алгоритм широко применяется в рекомендательных системах, только для разложение не матрицы *термин-документ*, а матрицы *пользователь-продукт* (*user-item matrix*)
- С помощью SVD-разложения мы выявляем скрытые интересы пользователя и заполняем нули в исходной матрице предсказаниями, которые говорят о том, насколько данный продукт может быть интересен данному пользователю



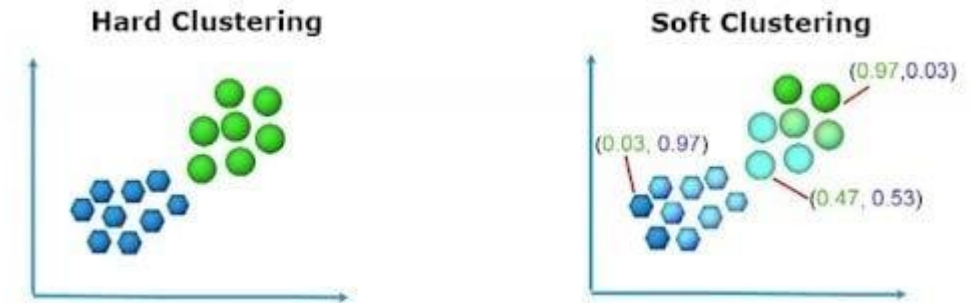
Матрица *пользователь-продукт*, в роли продуктов в данном случае выступают фильмы.

Мы знаем, что какие фильмы понравились пользователям, и хотим предсказать, насколько им понравятся фильмы, которые они еще не видели.

(картинка из [поста](#))

Но почему все это работает?

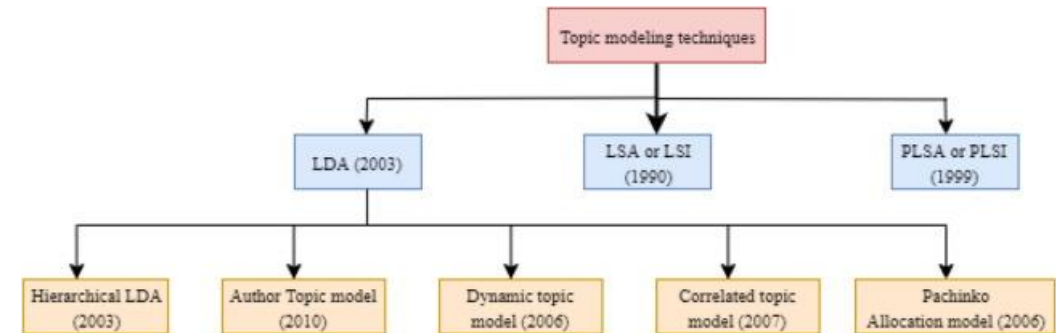
- Все, что мы до сих пор делали – это простые математические операции над матрицами
- Почему это приводит к тому, что у нас «выпадает в осадок» семантика? Почему компоненты dense эмбеддингов соответствуют тематикам?
- Одно из возможных объяснений – произошла мягкая кластеризация (soft clustering) по кластерам-тематикам
- Сжимая разреженные векторы в пространство низкой размерности, мы сближаем близкие по тематикам документы друг с другом так, что компоненты этих векторов начинают отражать сродство данного документа к кластерам-тематикам



Мягкая кластеризация: каждый объект получает некую меру близости к каждому из кластеров (в данном примере у нас 2 кластера)

Тематическое моделирование

- Идея моделирования «взаимодействия» документов, терминов и тематик получила развитие в таких классических методах как *pLSA* ([Hofmann, 1999](#)) или *LDA* ([Blei, 2003](#))
- Мы уже не разлагаем матрицы, а обучаем вероятностные генеративные модели со скрытыми тематиками, которые генерируют «наблюдаемые» нами документы
- Вся совокупность методов (включая *LSA*) называется тематическим моделированием (*topic modeling*, [Wikipedia](#))
- Современное тематическое моделирование, конечно же, с успехом использует глубокие нейронные сети



Классические алгоритмы тематического моделирования

Контексты

- Еще раз вспомним дистрибутивную гипотезу
- Слова обретают смысл когда встречаются в одинаковых контекстах
- Документы – это тоже контексты
- Когда мы работали с матрицей термин-документ, мы работали с контекстами!
- В качестве контекстов, при желании, можно использовать предложения, или скользящие окна, и т.п.
- Эти идеи можно развивать

■ : Center Word

■ : Context Word

c=0 The cute **cat** jumps over the lazy dog.

c=1 The **cute** **cat** **jumps** over the lazy dog.

c=2 **The** **cute** **cat** **jumps** **over** the lazy dog.

Примеры простых контекстов длиной c=0,1,2... вокруг слова CAT.

Картинка из [документации](#) к фреймворку Chainer

Word2Vec

- Можно зайти со стороны ML и, вместо «простого» разложения матриц, пытаться предсказывать слова по контекстам (и наоборот)
- Самая известная модель – *Word2Vec* (и ее более современный вариант *FastText*)
- Оказывается, *Word2Vec* тоже можно понимать как матричное разложение! (*)
- Про все поговорим в первой лекции про Нейропоиск

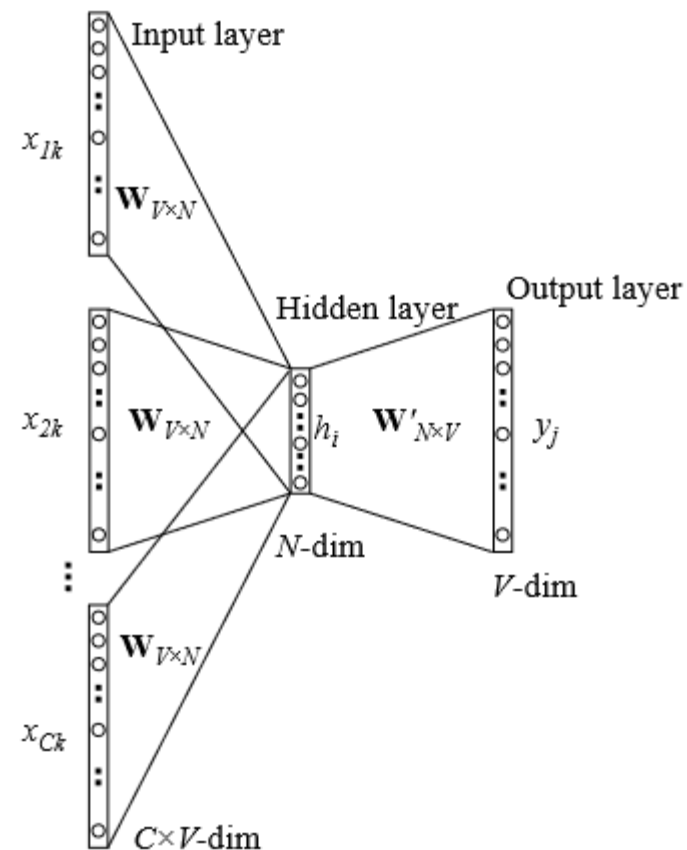


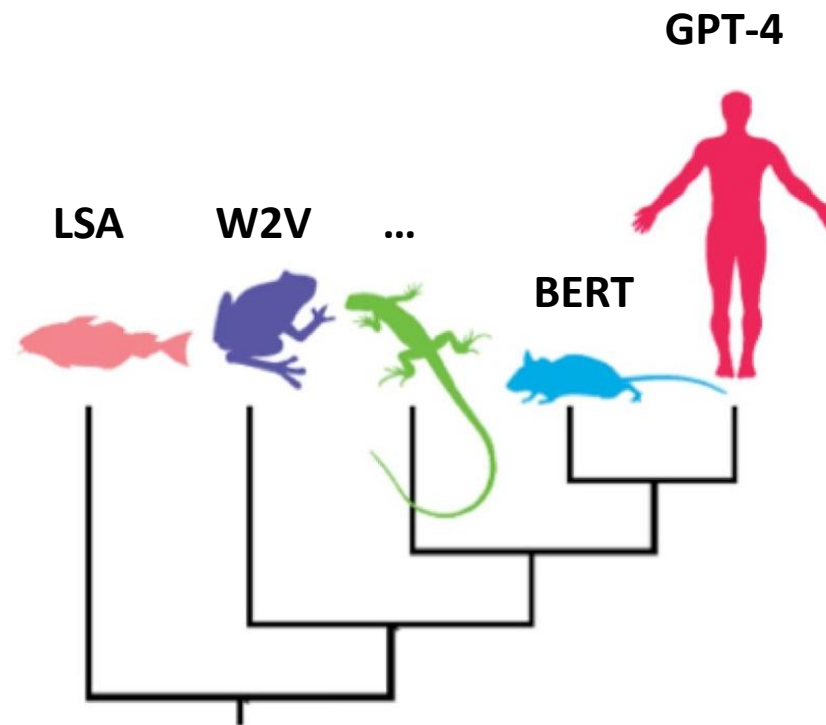
Figure 2: Continuous bag-of-word model

Модель word2vec CBOW (из [статьи](#) by Xin Rong, 2016)

(*) Levy, Omer, and Yoav Goldberg. "Neural word embedding as implicit matrix factorization." *Advances in neural information processing systems* 27 (2014).

Эволюция идей

- LSA – первая ступенька на пути к «пониманию»
- Все современное NLP, по сути, является развитием идей дистрибутивной семантики



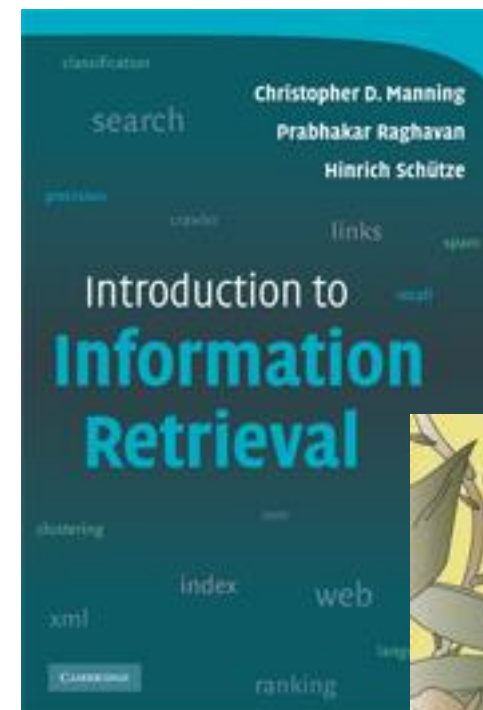
Дистрибутивная гипотеза + предсказание слов по контекстам

Материалы



Книги и статьи

- Книга Маннинга: Manning, Christopher D. *An introduction to information retrieval*. 2009. ([home](#))
 - Глава 6 про модель векторного пространства и TF-IDF
 - Глава 18 про латентно-семантический анализ
- Книга Берри и Брауна: Berry, Michael W., and Murray Browne. *Understanding search engines: mathematical modeling and text retrieval*. Society for Industrial and Applied mathematics, 2005. ([home](#))
 - Глава 3 про модель векторного пространства и TF-IDF
 - Глава 4 про латентно-семантический анализ (на примере не только SVD, но и QR-разложения)
- Замечательная статья про LSA/LSI: Deerwester, Scott, Susan T. Dumais, George W. Furnas, Thomas K. Landauer, and Richard Harshman. "Indexing by latent semantic analysis." (1990) (<https://psychology.uwo.ca/faculty/harshman/latentsa.pdf>)



Семинар и ДЗ



Семинар

- Разберемся как применять все это на практике
- Научимся вычислять TF-IDF-скоры близости текстов
- Применим векторы TF-IDF в задаче ранжирования
- Обучим семантические эмбединги по методу LSA, и, с их помощью, решим задачу семантической близости

ДЗ

- Сегодня не будет!

Спасибо за
внимание!

