Модели информационного поиска

Модели поиска

- Математическая модель – для осуществления процесса поиска
  - Предположения о релевантности – в математической форме

- Самая простая модель поиска
  - булева модель
  - Основная модель поиска 60-80-е

Булев поиск

- Два возможных результата для сопоставления запроса и документа
  - TRUE и FALSE
  - Поиск по полному совпадению
  - Простейшая форма ранжирования
- Запрос может специфицироваться посредством Булевых операторов
  - AND, OR, NOT
  - Могут быть использоваться операторы близости (proximity)

Булев поиск: пример

Какие пьесы Шекспира содержат слова
Brutus *AND* ***Caesar*** but *NOT* ***Calpurnia***?

Просмотр и сканирование пьес, чтобы найти ***Brutus*** and ***Caesar,*** и вычеркнуть те пьесы, где есть ***Calpurnia***?

Ответ: нет, так нельзя. Почему?
Очень медленно (для большой коллекции)

Чтобы избежать онлайнового просмотра документов,
нужно их заранее проиндексировать

Матрица терм-документ

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

1 if play contains word, 0 otherwise

Brutus *AND* ***Caesar*** *BUT NOT* ***Calpurnia***

Вектор вхождения

Вектор из {0,1} вхождений

Чтобы ответить на запроса:

Берем вектора ***Brutus, Caesar*** and ***Calpurnia*** (инверсия) ➔ и выполняем побитовую операцию *AND*.

110100 *AND* 110111 *AND* 101111 = 100100.

Ответы к запросу

Antony and Cleopatra, Act III, Scene ii

Agrippa [Aside to DOMITIUS ENOBARBUS]: Why, Enobarbus,

When Antony found Julius ***Caesar*** dead,

He cried almost to roaring; and he wept

When at Philippi he found ***Brutus*** slain.

Hamlet, Act III, Scene ii

Lord Polonius: I did enact Julius ***Caesar*** I was killed i' the

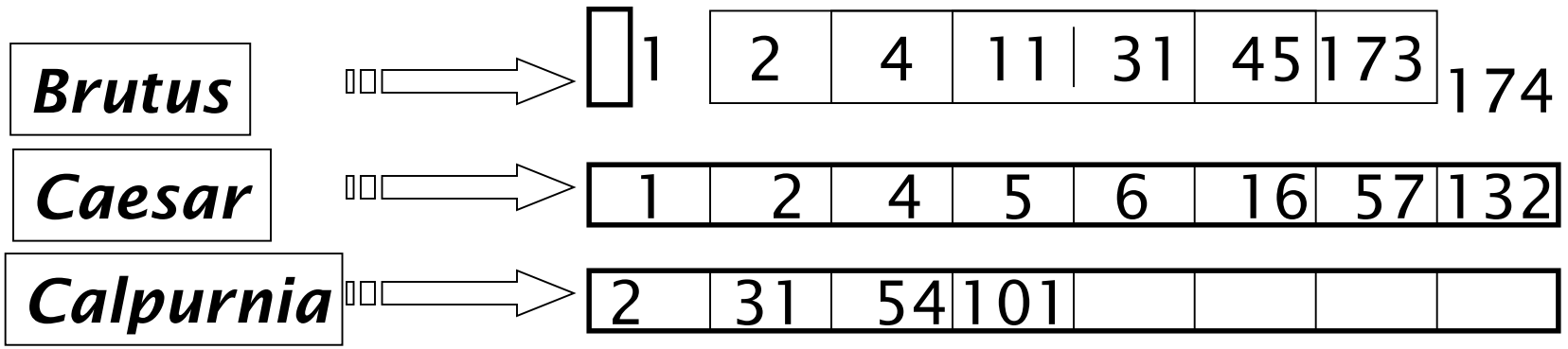
Capitol; ***Brutus*** killed me.

Большие коллекции

- Коллекция *N* = 1 миллион документов, каждый документ – около 100 слов
- Например, *M* = 500K отдельных термов
- Матрица терм – документ: 500K x 1M
  - Занимала бы сверх большие объемы
  - Очень разреженная
  - Поэтому другая форма представления

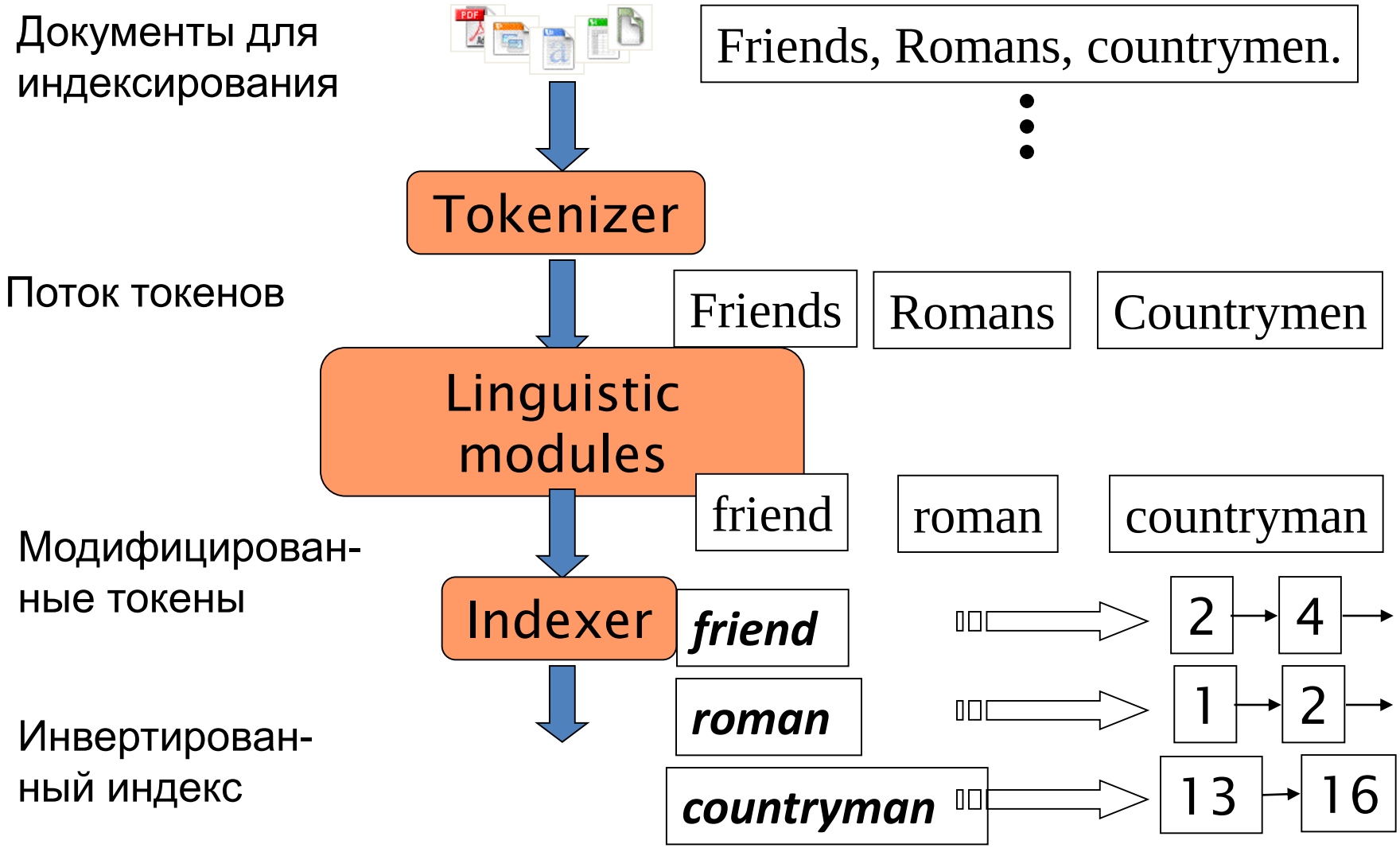
Инвертированный индекс

- Для каждого терма *t нужно хранить список документов, которые содержат t*.
  - Каждый документ идентифицируется номером документа - **docID**



Добавление слова ***Caesar*** к документу 14?

Создание инвертированного индекса



Шаги индексатора: Последовательность токенов

- Последовательность пар (Токен, DocID)

Doc 1

I did enact Julius Caesar I was killed i' the Capitol; Brutus killed me.

Doc 2

So let it be with Caesar. The noble Brutus hath told you Caesar was ambitious

Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Шаги индексатора: Сортировка

- Сортировка по токенам и затем DocId

Основной шаг индексирования

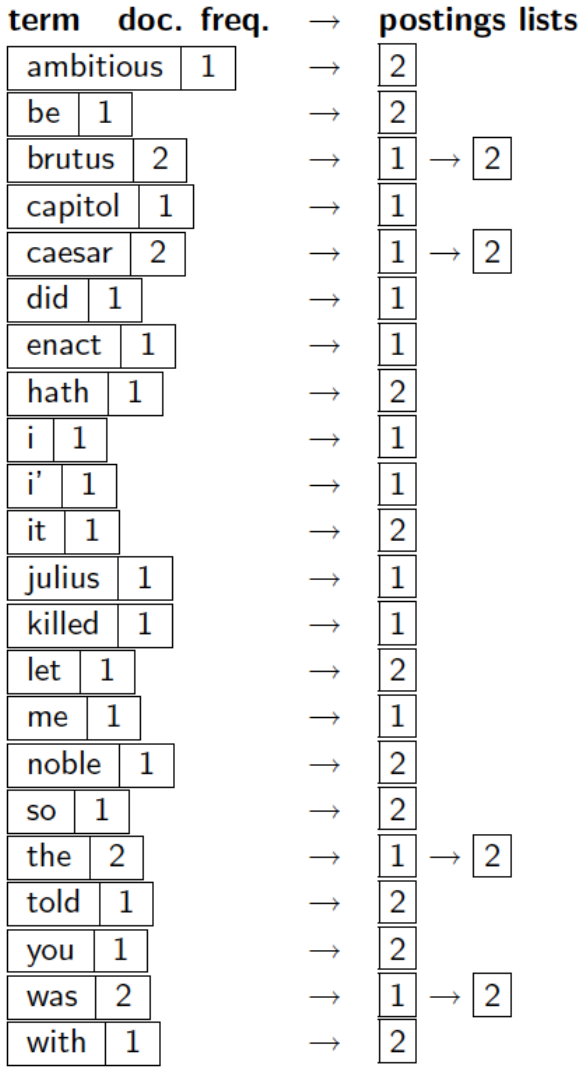
Term	docID
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
i'	1
the	1
capitol	1
brutus	1
killed	1
me	1
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
noble	2
brutus	2
hath	2
told	2
you	2
caesar	2
was	2
ambitious	2

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2

Шаги индексатора: Словарь & записи

- Вхождения токена в документе склеиваются
- Расчленяются на словарь и записи
- Добавляется информация о подокументной частотности.

Term	docID
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1
let	2
me	1
noble	2
so	2
the	1
the	2
told	2
you	2
was	1
was	2
with	2



Обработка запроса: AND

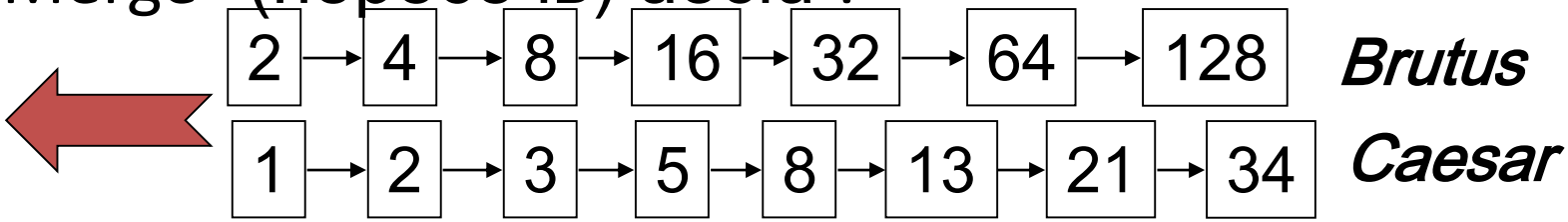
- Запрос:

  Brutus *AND* ***Caesar***

  – Найти ***Brutus*** в словаре;
    - Извлечь все его docid.

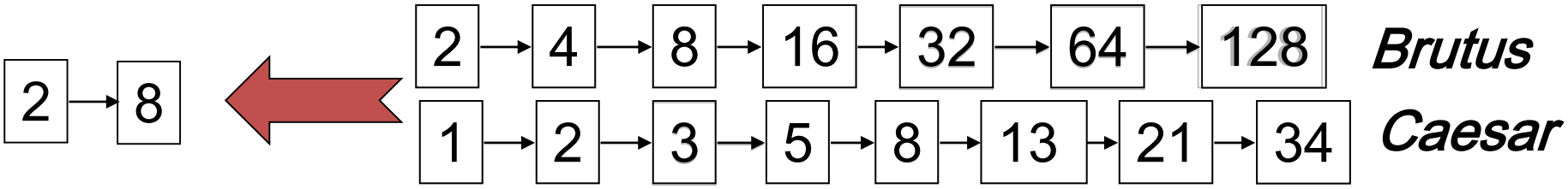
  – Найти *Caesar* в словаре;
    - Извлечь все его docid.

  – “Merge” (пересечь) docid :



Пересечение списков

- Одновременный проход по спискам документов по времени линейный от количества элементов в списках



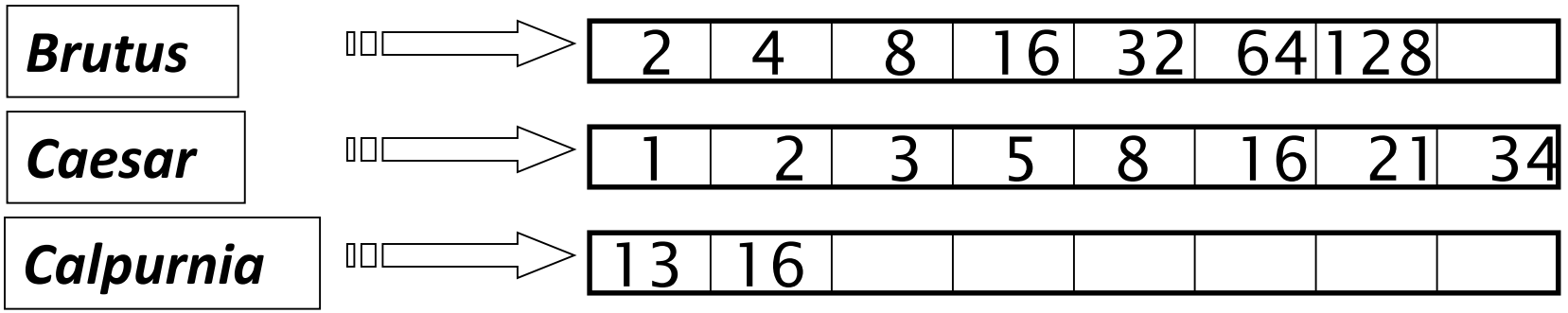
Если длины списков *x* и *y*, то пересечение O(*x*+*y*) операций
Необходимо: записи в списке должны быть отсортированы по docId

Пересечение двух списков docId (a “merge” algorithm)

```
INTERSECT(p1, p2)
 1  answer ← ⟨ ⟩
 2  while p1 ≠ NIL and p2 ≠ NIL
 3  do if docID(p1) = docID(p2)
 4        then ADD(answer, docID(p1))
 5             p1 ← next(p1)
 6             p2 ← next(p2)
 7        else if docID(p1) < docID(p2)
 8                then p1 ← next(p1)
 9                else p2 ← next(p2)
10  return answer
```

Оптимизация обработки запросов

- Лучший порядок для обработки запросов?
- Например, запрос имеет вид *AND* *n* термов.
- Для каждого из *n* термов, получить его список документов, и пересечь их.

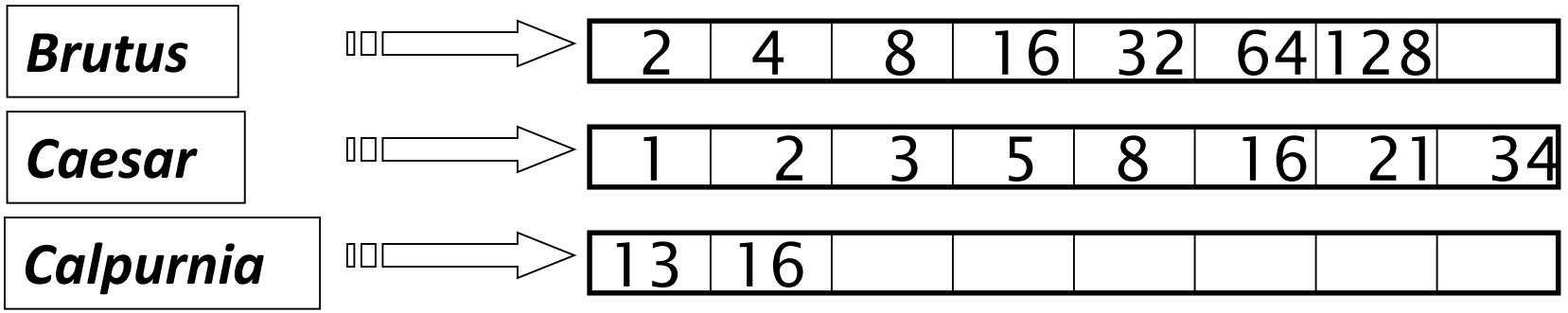


Query: ***Brutus*** *AND* ***Calpurnia*** *AND* ***Caesar***

Пример оптимизации обработки запросов

- Обработка должна происходить по мере увеличения частот:
  - *Начинаем с минимальных списков.*

Для этого нужно хранить количество документов в словаре



Выполняем запрос как (***Calpurnia*** *AND* ***Brutus)*** *AND* ***Caesar***.

Более общий случай оптимизации

- *(**madding** OR **crowd**) AND (**ignoble** OR **strife**)*
- Получить doc. freq для всех термов.
- Оценить размер каждого *OR* используя сумму его документных частот.
- Обрабатывать в процессе увеличения количества документов в OR.

Пример

- Какой порядок обработки запроса?

***(tangerine** OR **trees)** AND*
***(marmalade** OR **skies)** AND*
(kaleidoscope** OR **eyes)

Term	Freq
eyes	213312
kaleidoscope	87009
marmalade	107913
skies	271658
tangerine	46653
trees	316812

Булевские запросы: точное сопоставление

Булевская модель рассматривает запрос как булевское выражение

Булевские запросы используют *AND, OR* и *NOT связки между термами запроса*

- Каждый документ - набор термов
- Каждый запрос – по умолчанию операция И
- Точная выдача: Документ либо подходит к условию, либо не подходит.

Многие поисковые системы еще булевские:

Email, library catalog, Mac OS X Spotlight

Example: WestLaw

http://www.westlaw.com/

Largest commercial (paying subscribers) legal search service (started 1975; ranking added 1992)

Tens of terabytes of data; 700,000 users

Majority of users *still* use boolean queries

Example query:

What is the statute of limitations in cases involving the federal tort claims act?

LIMIT! /3 STATUTE ACTION /S FEDERAL /2 TORT /3 CLAIM

/3 = within 3 words, /S = in same sentence

Example: WestLaw

Another example query:

> *Requirements for disabled people to be able to access a workplace*
>
> disabl! /p access! /s work-site work-place (employment /3 place)

Note that SPACE is disjunction, not conjunction!

Long, precise queries; proximity operators; incrementally developed; not like web search

Many professional searchers still like Boolean search

> You know exactly what you are getting

But that doesn't mean it actually works better….

Булевский поиск

- Преимущества
  - Результаты предсказуемы, их легко объяснить
  - Могут быть встроены многие различные признаки
  - Эффективная обработка
- Недостатки
  - Качество выдачи зависит исключительно от пользователя
  - Простые запросы дают слишком много документов (нет упорядочения)
  - Длинные запросы сложно составить

Поиск, ведомый числом документов

- Последовательность запросов, направляемая числом документов
  - “lincoln” в новостных статьях
  - president AND lincoln
  - president AND lincoln AND NOT (automobile OR car)
  - president AND lincoln AND biography AND life AND birthplace AND gettysburg AND NOT (automobile OR car)
  - president AND lincoln AND (biography OR life OR birthplace OR gettysburg) AND NOT (automobile OR car)

Векторная модель информационного поиска

Ранжированный поиск

Thus far, our queries have all been Boolean.

- Documents either match or don't.

Good for expert users with precise understanding of their needs and the collection.

Not good for the majority of users.

- Most users incapable of writing Boolean queries (or they are, but they think it's too much work).
- Most users don't want to wade through 1000s of results.
  - This is particularly true of web search.

Ранжированные модели поиска

Rather than a set of documents satisfying a query expression, in ranked retrieval, the system returns an ordering over the (top) documents in the collection for a query

Free text queries: Rather than a query language of operators and expressions, the user's query is just one or more words in a human language

In principle, there are two separate choices here, but in practice, ranked retrieval has normally been associated with free text queries and vice versa

Ранжированные модели
vs. Булевские модели

When a system produces a ranked result set, large result sets are not an issue

- Indeed, the size of the result set is not an issue
- We just show the top *k* (≈ 10) results
- We don’t overwhelm the user

Premise: the ranking algorithm works

Основа ранжированного поиска – числовой вес

We wish to return in order the documents most likely to be useful to the searcher

How can we rank-order the documents in the collection with respect to a query?

Assign a score – say in [0, 1] – to each document

This score measures how well document and query “match”.

Сравнение запроса и документа

We need a way of assigning a score to a query/document pair

Let's start with a one-term query

If the query term does not occur in the document: score should be 0

The more frequent the query term in the document, the higher the score (should be)

We will look at a number of alternatives for this.

Попробуем: Коэффициент Жаккара

A commonly used measure of overlap of two sets *A* and *B*

jaccard*(A,B)* = $|A \cap B| \; / \; |A \cup B|$

jaccard*(A,A)* = 1

jaccard*(A,B)* = 0 if $A \cap B = 0$

A and *B* don't have to be the same size.

Always assigns a number between 0 and 1.

Проблемы с весом по Жаккару

It doesn’t consider *term frequency* (how many times a term occurs in a document)

Rare terms in a collection are more informative than frequent terms. Jaccard doesn’t consider this information

We need a more sophisticated way of normalizing for length

Булевская модель: бинарная матрица терм-документ

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Each document is represented by a binary vector $\in \{0,1\}^{|V|}$

Матрица частоты употребления терма в документе

Рассмотрим число вхождений терма в документ

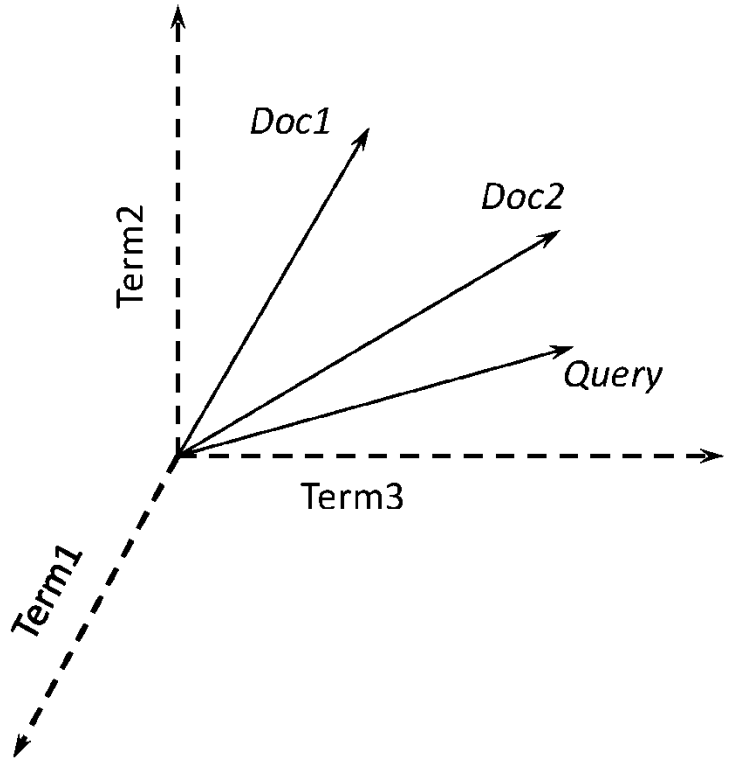
(=частота=frequency)

Каждый документ – это вектор частот $\mathbb{N}^v$

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	157	73	0	0	0	0
Brutus	4	157	0	1	0	0
Caesar	232	227	0	2	1	1
Calpurnia	0	10	0	0	0	0
Cleopatra	57	0	0	0	0	0
mercy	2	0	3	5	5	1
worser	2	0	1	1	1	0

Векторная модель

- 3-d pictures useful, but can be misleading for high-dimensional space



Модель «мешок слов»

Vector representation doesn't consider the ordering of words in a document

John is quicker than Mary and *Mary is quicker than John* have the same vectors

This is called the bag of words model.

The positional index was able to distinguish these two documents.

For now: bag of words model

Частотность по документам

Rare terms are more informative than frequent terms

Recall stop words

Consider a term in the query that is rare in the collection (e.g., *arachnocentric*)

A document containing this term is very likely to be relevant to the query *arachnocentric*

→ We want a high weight for rare terms like *arachnocentric*.

Подокументная частотность

Frequent terms are less informative than rare terms

Consider a query term that is frequent in the collection (e.g., *high, increase, line*)

A document containing such a term is more likely to be relevant than a document that doesn't

But it's not a sure indicator of relevance.

→ For frequent terms, we want high positive weights for words like *high, increase, and line*

But lower weights than for rare terms.

We will use document frequency (df) to capture this.

idf вес

df_t is the document frequency of t: the number of documents that contain t

- df_t is an inverse measure of the informativeness of t
- $df_t \leq N$

We define the idf (inverse document frequency) of t by

$$idf_t = \log_{10}(N/df_t)$$

We use log (N/df_t) instead of N/df_t to "dampen" the effect of idf.

idf example, suppose N = 1 million

term	df_t	idf_t
calpurnia	1	6
animal	100	4
sunday	1,000	
fly	10,000	2
under	100,000	1
the	1,000,000	0

$$idf_t = \log_{10}(N/df_t)$$

There is one idf value for each term t in a collection.

Воздействие idf на ранжирование

Does idf have an effect on ranking for one-term queries, like

- iPhone

idf has no effect on ranking one term queries

- idf affects the ranking of documents for queries with at least two terms
- For the query capricious person, idf weighting makes occurrences of capricious count for much more in the final document ranking than occurrences of person.

Частотность vs. Документная частотность

- The collection frequency of *t* is the number of occurrences of *t* in the collection, counting multiple occurrences.
- Example:

Word	Collection frequency	Document frequency
insurance	10440	3997
try	10422	8760

- Which word is a better search term (and should get a higher weight)?

tf-idf weighting

- The tf-idf weight of a term is the product of its tf weight and its idf weight.

- Best known weighting scheme in information retrieval
  - Note: the "-" in tf-idf is a hyphen, not a minus sign!
  - Alternative names: tf.idf, tf x idf
- Scheme (1+log (tf)) is often used
- Increases with the number of occurrences within a document
- Increases with the rarity of the term in the collection

Binary → count → weight matrix

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5,25	3,18	0	0	0	0,35
Brutus	1,21	6,1	0	1	0	0
Caesar	8,59	2,54	0	1,51	0,25	0
Calpurnia	0	1,54	0	0	0	0
Cleopatra	2,85	0	0	0	0	0
mercy	1,51	0	1,9	0,12	5,25	0,88
worser	1,37	0	0,11	4,15	0,25	1,95

Each document is now represented by a real-valued vector of tf-idf weights $\in \mathbb{R}^{|V|}$

Документы как векторы

So we have a |V|-dimensional vector space

Terms are axes of the space

Documents are points or vectors in this space

Very high-dimensional: tens of millions of dimensions when you apply this to a web search engine

These are very sparse vectors - most entries are zero.

Формализация близости векторов

First cut: distance between two points

(= distance between the end points of the two vectors)

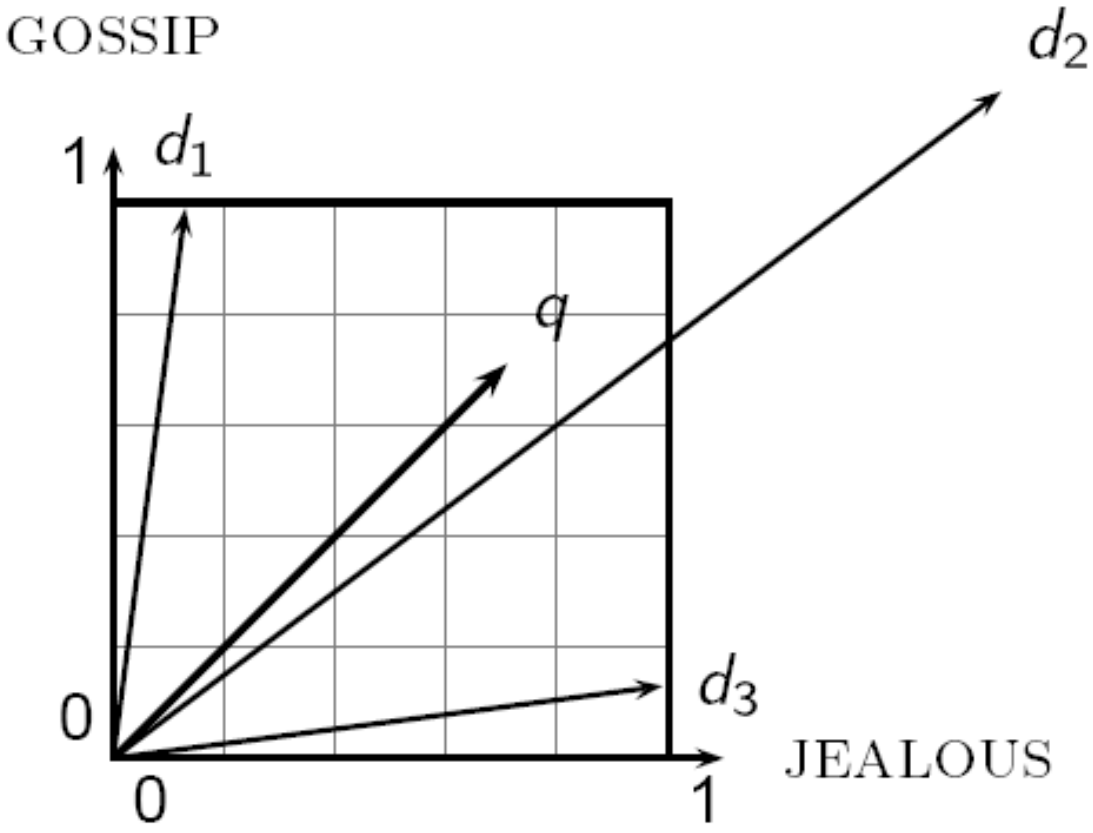
Euclidean distance?

Euclidean distance is a bad idea . . .

. . . because Euclidean distance is large for vectors of different lengths.

Why distance is a bad idea

The Euclidean distance between $\vec{q}$ and $\vec{d_2}$ is large even though the distribution of terms in the query $\vec{q}$ and the distribution of terms in the document $\vec{d_2}$ are very similar.



Использование угла вместо расстояния

Thought experiment: take a document *d* and append it to itself. Call this document *d*′.

“Semantically” d and d′ have the same content

The Euclidean distance between the two documents can be quite large

The angle between the two documents is 0, corresponding to maximal similarity.

От углов к косинусам

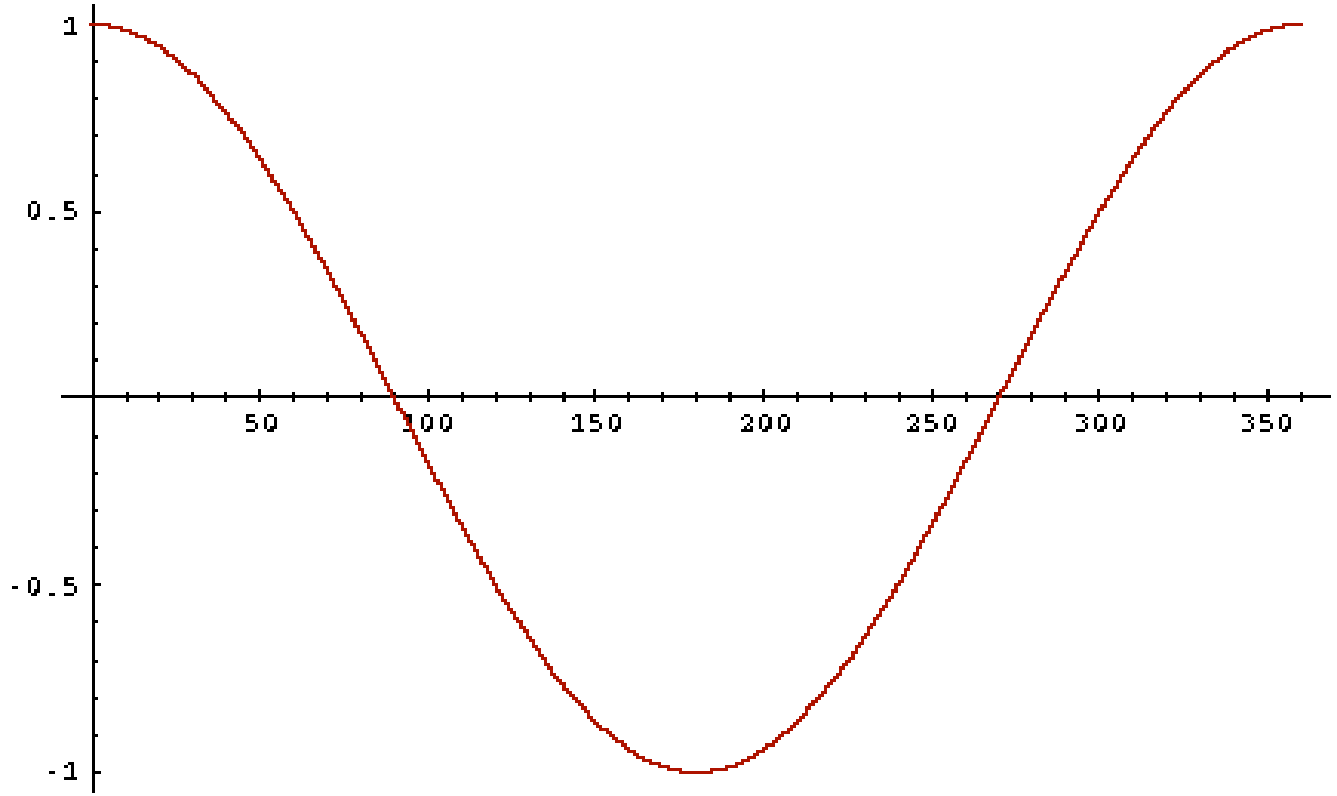
The following two notions are equivalent.

Rank documents in decreasing order of the angle between query and document

Rank documents in increasing order of cosine(query,document)

Cosine is a monotonically decreasing function for the interval [0^o, 180^o]

От углов к косинусам



But how – *and why* – should we be computing cosines?

Нормализация длины вектора

A vector can be (length-) normalized by dividing each of its components by its length – for this we use the L_2 norm:

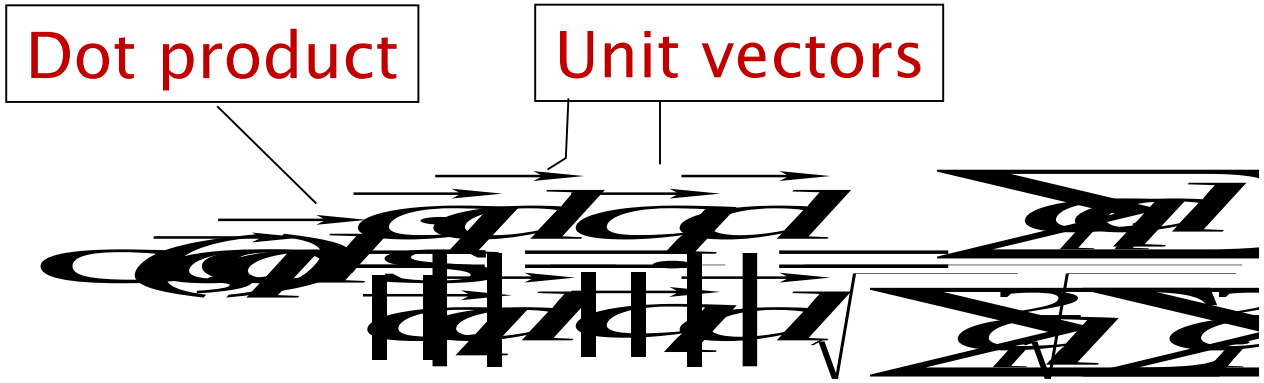
$$\|\vec{x}\|_2 = \sqrt{\sum_i x_i^2}$$

Dividing a vector by its L_2 norm makes it a unit (length) vector (on surface of unit hypersphere)

Effect on the two documents d and d′ (d appended to itself) from earlier slide: they have identical vectors after length-normalization.

Long and short documents now have comparable weights

Косинус (запрос,документ)

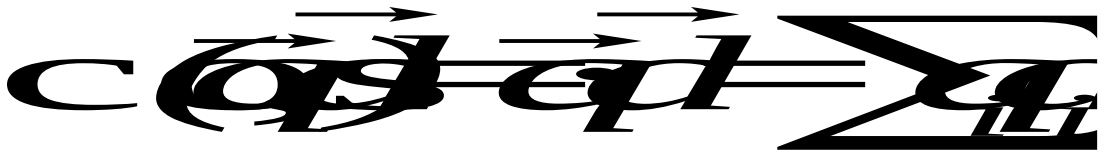


q_i is the tf-idf weight of term i in the query
d_i is the tf-idf weight of term i in the document

$\cos(\vec{q},\vec{d})$ is the cosine similarity of $\vec{q}$ and $\vec{d}$ … or, equivalently, the cosine of the angle between $\vec{q}$ and $\vec{d}$.

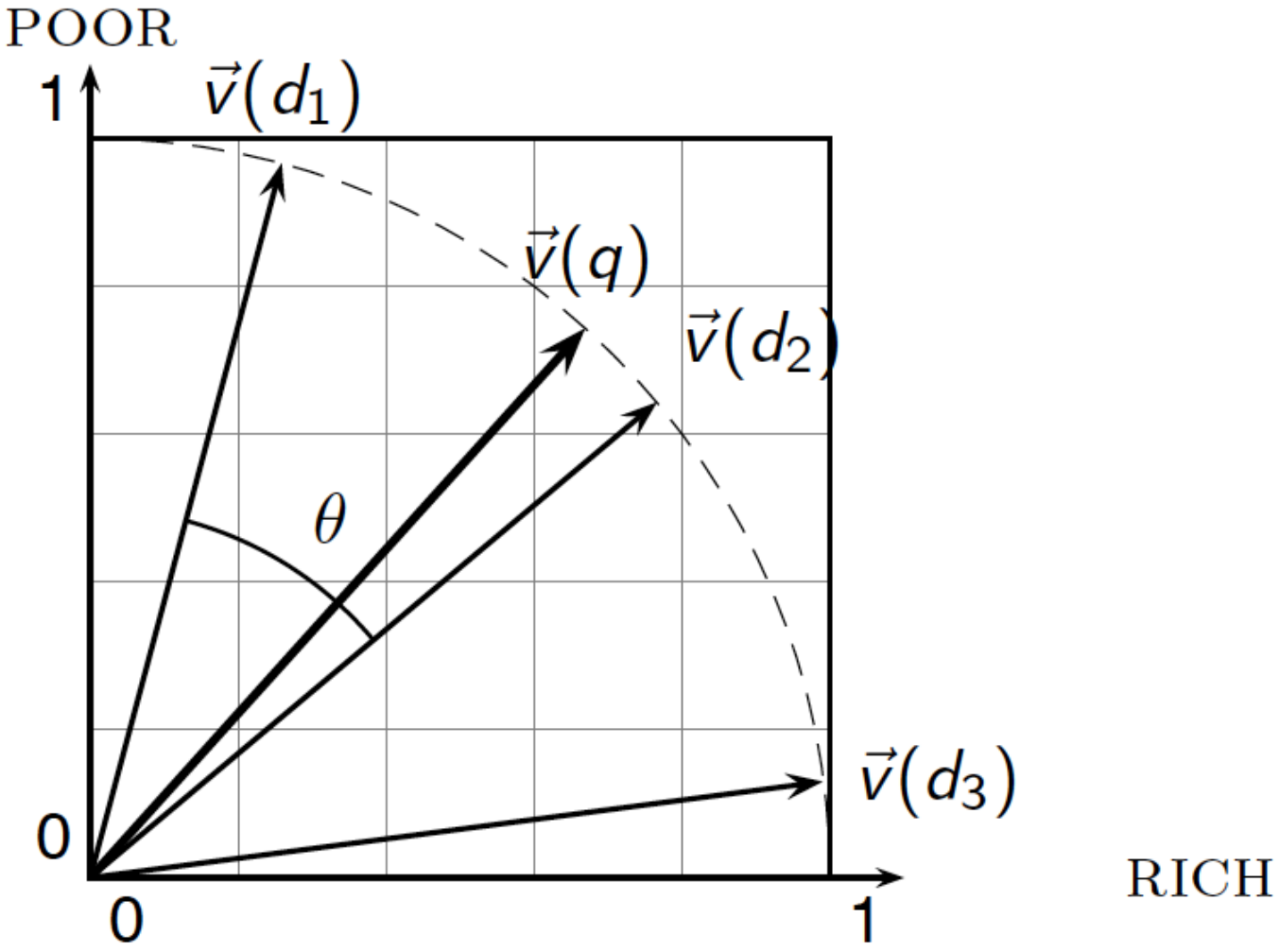
Косинус для нормализованных векторов

For length-normalized vectors, cosine similarity is simply the dot product (or scalar product):



for q, d length-normalized.

Cosine similarity illustrated



Варианты для tf-idf

Term frequency		Document frequency		Normalization	
n (natural)	$\mathrm{tf}_{t,d}$	n (no)	1	n (none)	1
l (logarithm)	$1 + \log(\mathrm{tf}_{t,d})$	t (idf)	$\log \frac{N}{\mathrm{df}_t}$	c (cosine)	$\frac{1}{\sqrt{w_1^2+w_2^2+\ldots+w_M^2}}$
a (augmented)	$0.5 + \frac{0.5 \times \mathrm{tf}_{t,d}}{\max_t(\mathrm{tf}_{t,d})}$	p (prob idf)	$\max\{0, \log \frac{N-\mathrm{df}_t}{\mathrm{df}_t}\}$	u (pivoted unique)	$1/u$
b (boolean)	$\begin{cases} 1 & \text{if } \mathrm{tf}_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$			b (byte size)	$1/CharLength^{\alpha}$, $\alpha < 1$
L (log ave)	$\frac{1+\log(\mathrm{tf}_{t,d})}{1+\log(\mathrm{ave}_{t \in d}(\mathrm{tf}_{t,d}))}$				

Columns headed ‘n’ are acronyms for weight schemes.

Why is the base of the log in idf immaterial?

Схема весов может различаться в Запросах и документах

Many search engines allow for different weightings for queries vs. documents

SMART Notation: denotes the combination in use in an engine, with the notation *ddd.qqq,* using the acronyms from the previous table

A very standard weighting scheme is: lnc.ltc

Document: logarithmic tf (l as first character), no idf and cosine normalization

Query: logarithmic tf (l in leftmost column), idf (t in second column), no normalization …

Заключение

Представление запроса как взвешенного tf-idf вектора

Представление документа как взвешенного tf-idf вектора

Вычисление косинусной меры между вектором запроса и вектором документа – вес для ранжирования

Ранжирование документов по мере снижения веса

Выдача первых *K* (e.g., *K* = 10) документов пользователю

Запрос: car insurance, схема ntc.nnn

Term	df	idf	d1	d2	d3
car	18165	1.65	27	4	24
auto	6723	2.08	3	33	0
Insu-rance	19241	1.62	0	33	29
best	25235	1.5	14	0	17