UNIVERSITÀ DEGLI STUDI DI MILANO

MEC ITALY

THE VECTOR SPACE MODEL

ALFIO FERRARA, STEFANO MONTANELLI

Department of Computer Science

Via Comelico 39, 20135 Milano

{alfio.ferrara,stefano.montanelli}@unimi.it
http://islab.di.unimi.it/

Outline

- Term weighting
  - Term frequency
  - Document frequency
  - Tf-Idf
  - Term specificity
- The vector space model
  - Cosine similarity
  - Query processing

RECALL OF THE MAIN FUNCTIONS FOR TERM FREQUENCY

Fig. 6.15, cap 6

n (natural)	$tf_{t,d}$ as the number of occurrences of t in d
l (logarithm)	$1 + \log(tf_{t,d})$
a (augmented)	$0.5 + \frac{0.5tf_{t,d}}{max_{t'}(tf_{t'd})}$
b (boolean)	1 if $tf_{t,d} > 0, 0$ otherwise
L (log avg)	$\frac{1+\log(tf_{t,d})}{1+\log(avg_{t\in d}(tf_{t,d}))}$
M (maximum tf norm)	$k + (1-k)\frac{tf_{t,d}}{tf_{max}(d)}$

Example (no stemming)

Bob Dylan, Mr. Tambourine Man, 1965 [ID: 913]

http://lyrics.wikia.com/Bob_Dylan:Mr._Tambourine_Man

token	natural	logarithm	augmented	log avg	max tf norm
m	12.0	3.485	1.0	2.188	0.486
tambourine	11.0	3.398	0.958	2.133	0.479
song	10.0	3.303	0.917	2.074	0.471
hey	10.0	3.303	0.917	2.074	0.471
mr	10.0	3.303	0.917	2.074	0.471
play	10.0	3.303	0.917	2.074	0.471
man	10.0	3.303	0.917	2.074	0.471
sleepy	5.0	2.609	0.708	1.638	0.436
ll	5.0	2.609	0.708	1.638	0.436
going	5.0	2.609	0.708	1.638	0.436

Example (Stemming)

Bob Dylan, Mr. Tambourine Man, 1965 [ID: 913]

http://lyrics.wikia.com/Bob_Dylan:Mr._Tambourine_Man

token	natural	logarithm	augmented	log avg	max tf norm
m	12.0	3.485	1.0	2.138	0.486
tambourin	11.0	3.398	0.958	2.084	0.479
song	10.0	3.303	0.917	2.026	0.471
hey	10.0	3.303	0.917	2.026	0.471
mr	10.0	3.303	0.917	2.026	0.471
play	10.0	3.303	0.917	2.026	0.471
man	10.0	3.303	0.917	2.026	0.471
go	7.0	2.946	0.792	1.807	0.45
jingle-jangl	5.0	2.609	0.708	1.601	0.436
sleepi	5.0	2.609	0.708	1.601	0.436

Document frequency

Besides term frequency, another important measure for weighting terms is the **document frequency**

The document frequency $df(t)$ of a term t is given by the number of documents d_i where $tf_{t,d_i} > 0$

In many applications, we are interested in terms that are specific of a small subset of documents and, thus, have a low document frequency

To this end, we define the **inverse document frequency** idf_t as

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

Example

token	df	idf
m	1295.0	1.203
tambourine	1.0	8.369
song	245.0	2.868
hey	191.0	3.117
mr	53.0	4.399
play	206.0	3.041
man	766.0	1.728
sleepy	11.0	5.971
ll	1173.0	1.302
going	262.0	2.801

TF-IDF

Combining **term frequency** and **inverse document frequency** we obtain a measure that takes into account the specific relevance of a terms with respect to a document

$$tfidf_{t,d} = tf_{t,d} \cdot idf_t$$

Such a measure is:

- high when t appears in a small number of documents (high discriminating)
- low when t occurs few times in d or occurs in many documents (generic)

Example (no stemming)

Bob Dylan, Mr. Tambourine Man, 1965 [ID: 913]

http://lyrics.wikia.com/Bob_Dylan:Mr._Tambourine_Man

token	tf	idf	tf-idf
tambourine	3.398	8.369	28.437
jingle-jangle	2.609	8.369	21.838
followin	2.609	7.676	20.029
sleepy	2.609	5.971	15.581
mr	3.303	4.399	14.527
hey	3.303	3.117	10.293
…	…		
man	3.303	1.728	5.706
…	…		
m	3.485	1.203	4.191

Example (no stemming)

F. De André, Bocca Di Rosa, 1967 [ID: 15]

http://lyrics.wikia.com/Fabrizio_De_Andr%C3%A9:Bocca_Di_Rosa

token	tf	idf	tf-idf
pennacchi	2.099	7.676	16.108
metteva	2.099	7.676	16.108
bocca	3.079	4.477	13.787
paesino	1.693	7.676	12.996
rosa	2.792	4.608	12.864
arrivarono	1.693	7.27	12.31
stazione	2.386	4.903	11.7
pretese	1.693	6.759	11.445
chiamavano	1.693	6.577	11.136
commissario	1.693	6.423	10.875

Specificity

Term specificity is based on the idea of evaluating the usage of a term in a corpus $\mathcal{C}$ with respect to a different corpus $\mathcal{D}$.

Given two corpora $\mathcal{C}$ and $\mathcal{D}$, we denote $f(t)$ the relative frequency of the term t in $\mathcal{C}$ and $f^*(t)$ the relative frequency of t in $\mathcal{D}$. Term specificity $spec(t)$ is calculated as:

$$spec(t) = \frac{f(t) - f^*(t)}{\sqrt{f^*(t)}}$$

Interpretation:

- $spec(t) > 0$: t is over-used in $\mathcal{C}$ wrt $\mathcal{D}$
- $spec(t) = 0$: t is equally-used in $\mathcal{C}$ and $\mathcal{D}$
- $spec(t) < 0$: t is over-used in $\mathcal{D}$ wrt $\mathcal{C}$

USAGE OF SPECIFICITY

Specificity is typically used for evaluating the terminology in a subset of the main corpus, with $\mathcal{C} \subset \mathcal{D}$.

The specificity of a document d_i can be evaluated by defining $\mathcal{C}$ as a set of documents containing only d_i.

$$spec(d_i) = \sum_{j=0}^{n} \frac{f(t_j) - f_*(t_j)}{\sqrt{f_*(t_j)}},$$

with n and the number of terms in d_i

Bolasco, Sergio. L'analisi automatica dei testi: fare ricerca con il text mining. Carocci Editore, 2013.

EXAMPLE: SPECIFICITY FROM TOKEN COLLECTIONS

```
def get_frequencies(self, query):
    match = {'$match': query}
    unwind = {'$unwind': "$tokens"}
    group = {'$group': {'_id': "$tokens", 'count': {'$sum': 1}}}
    stats = self.db[self.docs].aggregate([
        match,
        unwind,
        group
    ])
    stats = list(stats)
    total = sum([x['count'] for x in stats])
    freq = dict((x['_id'], float(x['count']) / total) for x in stats)
    return freq

def specificity(self, query):
    star = self.get_frequencies({})
    freq = self.get_frequencies(query)
    spec = {}
    for token, f in freq.items():
        spec[token] = (f - star[token]) / math.sqrt(star[token])
    return spec
```

EXAMPLE

{'artist': "De Andrè"}

Token	Spec
amore	0.179
ogni	0.128
senza	0.128
madre	0.126
occhi	0.123
...	...
lenta	0.05
atti	0.05
...	...
little	-0.053
gonna	-0.059

{title: "Help!"}

Token	Spec
help	4.707
younger	3.942
appreciate	3.377
self-assured	3.118
indepen	2.702
...	...
...	...

{year: {"\$gte": 1960, "\$lte": 1965}}

Token	Spec
amen	0.092
ooh-ooh	0.078
bye-bye	0.077
...	...
face	0.01
sent	0.01
...	...
jesus	-0.0
claus	-0.0
records	-0.0

Bag of Words

According to one of the previous weighting schemes, each document can be seen as a **bag of words** (no matter the order of words in the document), each associated with a corresponding weight.

Extending this idea, given any collection of terms T, a document can be represented as a collection of weights W over T, where each weight w_i is equal to 0 if the term $t_i \in T$ is not in the document or equal to the weight t_i has in the document otherwise.

Example:

['need', 'love', 'play']

Doc ID	Title	Weights
1161	Can't Buy Me Love	[0.071, 0.531, 0.0]
1159	All You Need Is Love	[0.58, 0.798, 0.01]
1167	Something	[0.085, 0.085, 0.0]
1171	That's All Right (Mama)	[0.036, 0.036, 0.0]
1095	I Should Have Known Better	[0.0, 0.615, 0.0]

Vector space model

Given a dictionary $\mathcal{D}$ of size D, we can define a multidimensional space with D dimensions, where each dimension i corresponds to the ith term in $\mathcal{D}$

In such a space, a document d is represented by a vector $\overrightarrow{V_d}$ in which each component i is the weight of the ith term of $\mathcal{D}$ in d (or 0)

Example on Two Dimensions

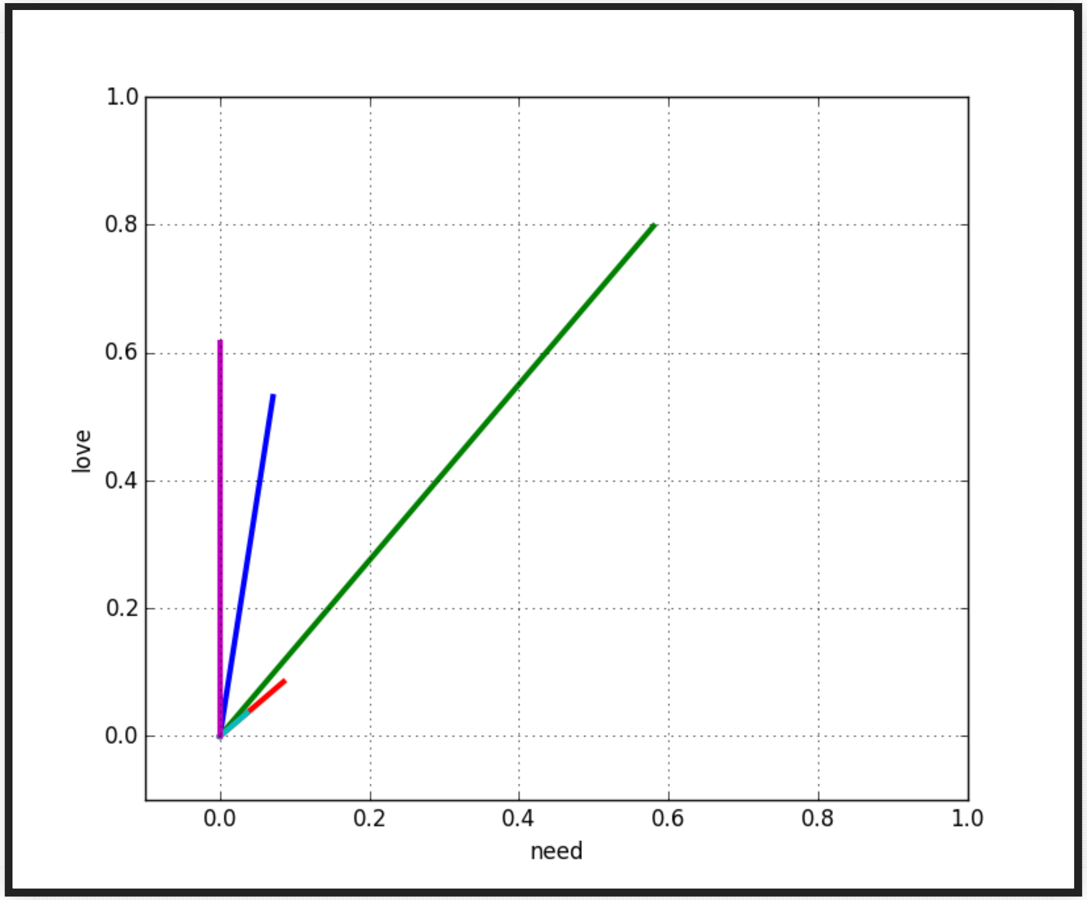
1161 Can't Buy Me Love
[0.071, 0.531]

1159 All You Need Is Love
[0.58, 0.798]

1167 Something
[0.085, 0.085]

1171 That's All Right (Mama)
[0.036, 0.036]

1095 I Should Have Known Better
[0.0, 0.615]



UNIT VECTORS

The weights of each vector are biased by the document length and the number of different terms in each document, resulting in vectors of different lengths

To overcome this problem, we normalize each document vector $\overrightarrow{V}$ in a **unit vector** $\overrightarrow{v}$ defined as

$$\overrightarrow{v} = \frac{\overrightarrow{V}}{\sqrt{\sum_{i=1}^{M} \overrightarrow{V}^2}}$$

,

where $\sqrt{\sum_{i=1}^{M} \overrightarrow{V}^2}$ is the Euclidean length of $\overrightarrow{V}$

EXAMPLE ON TWO DIMENSIONS

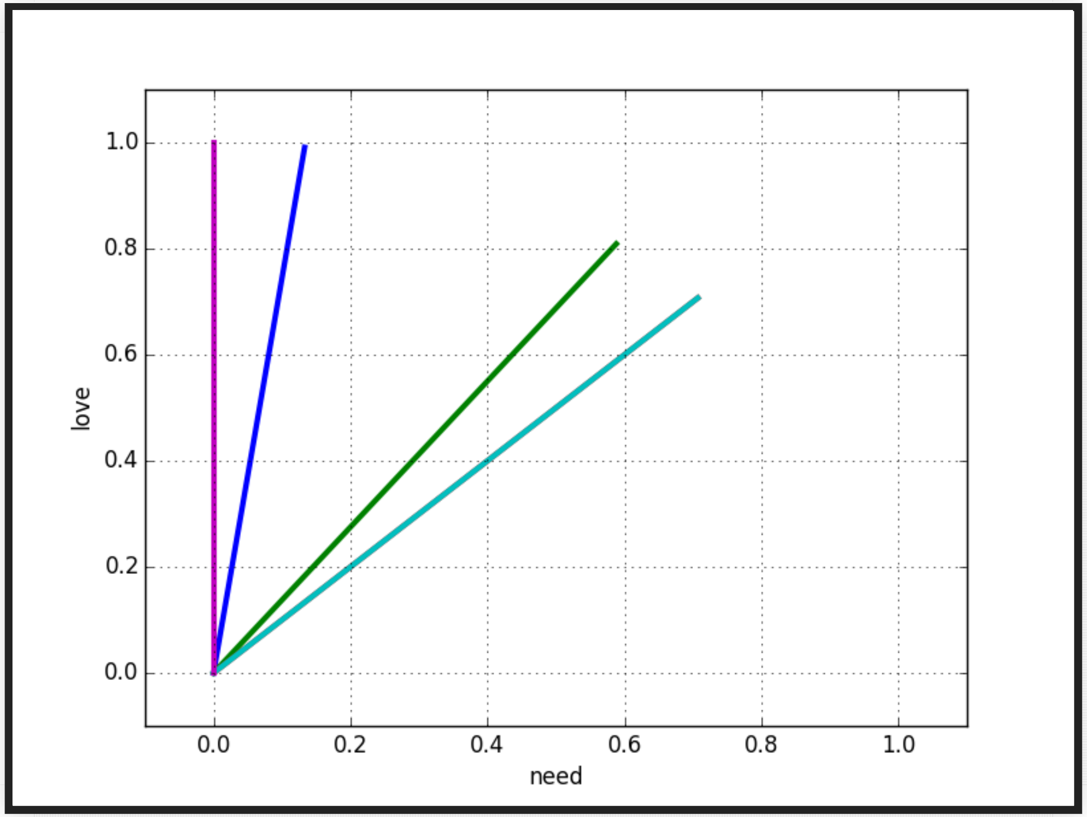
1161 Can't Buy Me Love
[0.133, 0.991]

1159 All You Need Is Love
[0.588, 0.809]

1167 Something
[0.707, 0.707]

1171 That's All Right (Mama)
[0.707, 0.707]

1095 I Should Have Known Better
[0.0, 1.0]



Example with songs

	love	re	need	oh	say	things	know	like	right	yeah
Can't Buy Me Love	0.83	0.0	0.11	0.5	0.17	0.11	0.0	0.0	0.11	0.0
All You Need Is Love	0.81	0.01	0.59	0.02	0.01	0.0	0.01	0.0	0.0	0.07
Something	0.13	0.13	0.13	0.0	0.0	0.13	0.96	0.13	0.0	0.0
That's All Right (Mama)	0.05	0.05	0.05	0.0	0.0	0.0	0.0	0.0	0.98	0.18
I Should Have Known Better	0.84	0.3	0.0	0.3	0.3	0.08	0.0	0.08	0.0	0.0

Cosine similarity

In the vector space model a natural function of similarity between documents is the cosine similarity of their vectors

$$sim(d_1, d_2) = \frac{\overrightarrow{V_{d_1}} \cdot \overrightarrow{V_{d_2}}}{\sqrt{\sum_{i=1}^{M} \overrightarrow{V_{d_1}}^2} \sqrt{\sum_{i=1}^{M} \overrightarrow{V_{d_2}}^2}} = \overrightarrow{v_{d_1}} \cdot \overrightarrow{v_{d_2}}$$

where the dot product $\overrightarrow{v_{d_1}} \cdot \overrightarrow{v_{d_2}}$ is defined as

$$\overrightarrow{v_{d_1}} \cdot \overrightarrow{v_{d_2}} = \sum_{i=1}^{M} v_i^{d_1} v_i^{d_2}$$

SCORE CALCULATION

```
def cosine_score(query, doc_index, docs):
    scores = {}
    length = {}
    for doc, data in docs.items():
        length[doc] = len(data[0])
    q_index = {}
    for q in query:
        try:
            q_index[q] += 1
        except KeyError:
            q_index[q] = 1.0
    e = euclidean_len(q_index.values())
    for q, wq in q_index.items():
        wq /= e
        if q in doc_index.keys():
            for posting in doc_index[q]:
                try:
                    scores[posting[0]] += wq * posting[1]
                except KeyError:
                    scores[posting[0]] = wq * posting[1]
    for d, l in scores.items():
```

Query processing

In the vector space model, for query processing we first transform the query in a vector over the vector model and then we calculate the cosine similarity (or the score function) on against the other documents

Example

match the song **All You Need Is Love**
answers
All You Need Is Love, Beatles 1.0
I Should Have Known Better, Beatles 0.509122694323
Can't Buy Me Love, Beatles 0.47381155953
Something, Beatles 0.123470630311
That's All Right (Mama), Beatles 0.0612707998333

Fast score function

Usually, queries are composed by a few terms, resulting in unit vectors that have very few non-zero components and where the non-zero components are equal

Now, for processing queries, we are interested in the relative, rather than absolute, scores of the documents in the collection

To this end, it suffices to compute the cosine similarity between each document unit vector $\vec{v}_d$ and the query vector $\vec{V}_q$ of the query, in which all the non-zero components of the query vector are set to 1

Example

```python
def fast_cosine_score(query, doc_index, docs):
    scores = {}
    length = {}
    for doc, data in docs.items():
        length[doc] = len(data[0])
    for q in query:
        if q in doc_index.keys():
            for posting in doc_index[q]:
                try:
                    scores[posting[0]] += posting[1]
                except KeyError:
                    scores[posting[0]] = posting[1]
    for d, l in scores.items():
        scores[d] = scores[d] / length[d]
    return sorted(scores.items(), key=lambda x: -x[1])
```

EXAMPLE MATCHING

Match with score function

match ['love', 'girl']

answers

All You Need Is Love, Beatles 0.47

I Should Have Known Better, Beatles 0.126

Can't Buy Me Love, Beatles 0.122

Something, Beatles 0.03

That's All Right (Mama), Beatles 0.019

Match with relative score

match ['love', 'girl']

answers

All You Need Is Love, Beatles 0.375

I Should Have Known Better, Beatles 0.166

Can't Buy Me Love, Beatles 0.137

That's All Right (Mama), Beatles 0.022

Something, Beatles 0.02

REDUCE TIME OF COMPUTATION

A general problem in using the vector space model in real systems is the time required for computing vector similarity given a query.

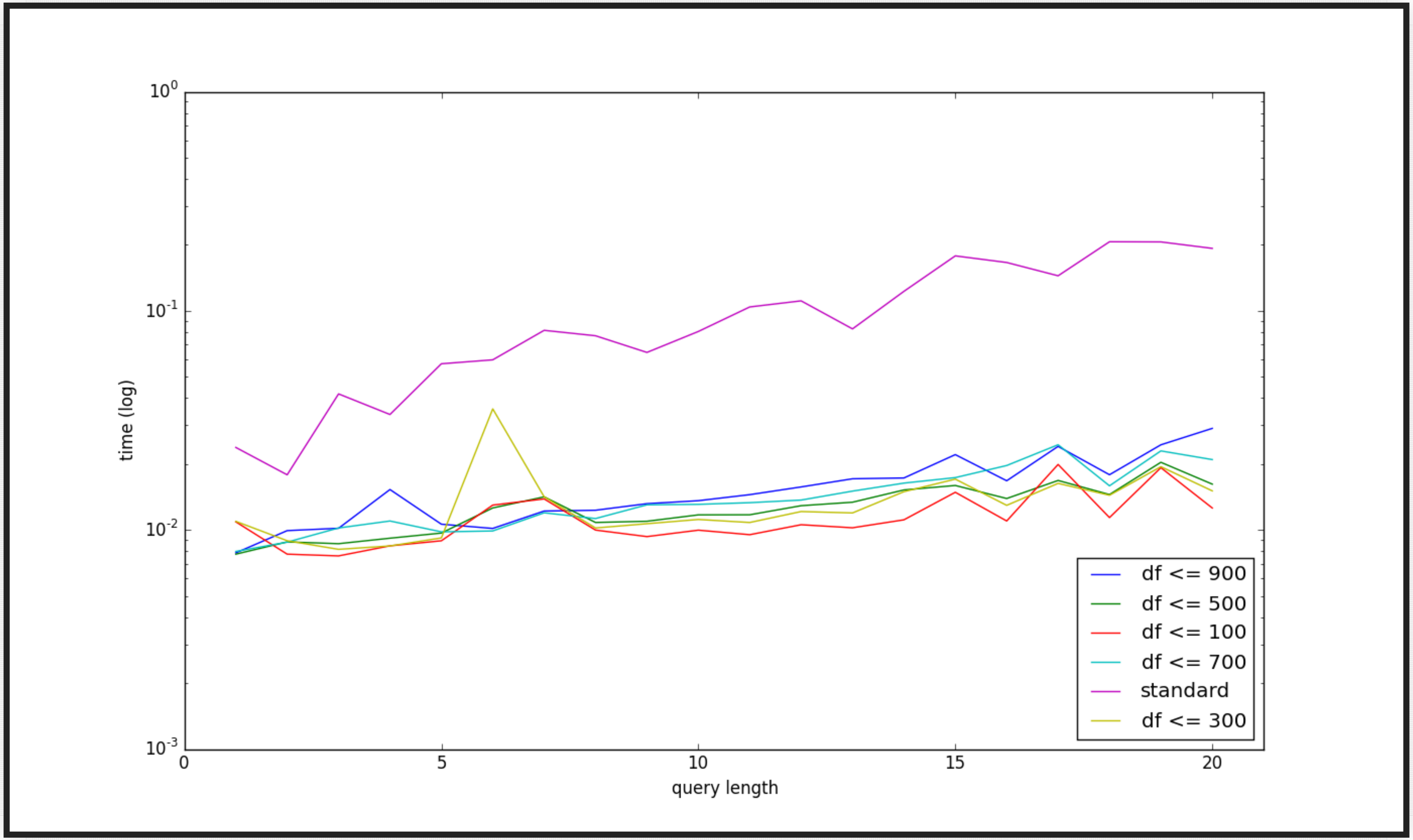
In the following, some heuristics will be presented to the aim of speeding up the **fast cosine score** function.

Index elimination

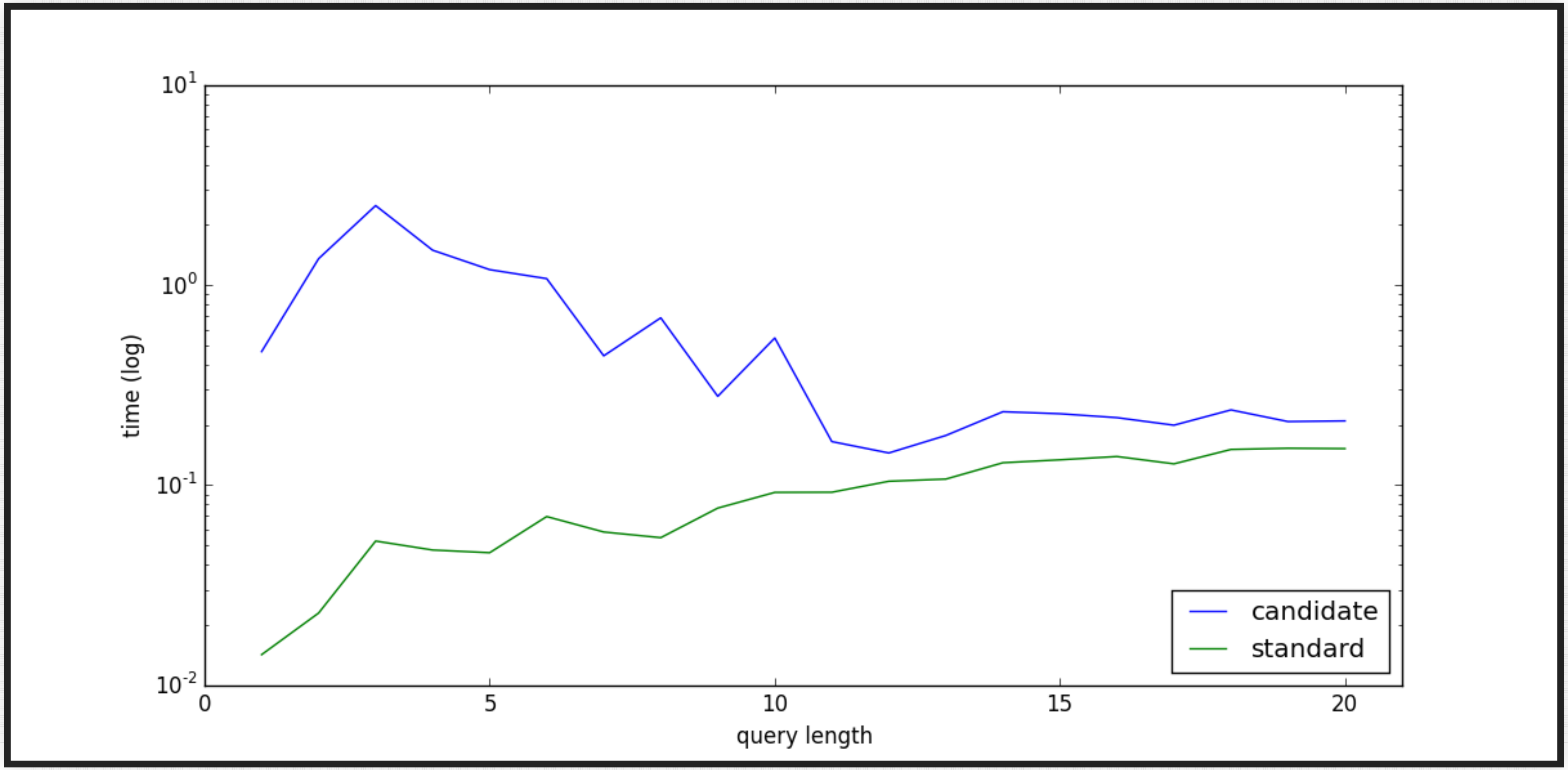
A common way of reducing computation, especially for long queries, is to reduce the number of index keys and/or postings that are taken into account

- **idf filter:** given a threshold th_{idf}, we skip all the query tokens qt such that $idf(qt) \leq th_{idf}$; this is like considering common terms as if they were stopwords.
- **terminology intersection:** we take into account only documents having a large token intersection with the query. This can be done as a pre-filter operation over the index, aggregating by document. This can lead to a small number of results.

EXAMPLE OF IDF OPTIMIZATION



EXAMPLE OF CANDIDATE OPTIMIZATION



Champion list

Precompute for each term t in the index the set C of the documents with the highest weights for t (e.g., the highest TF). C is called the **champion list** for t

At query time, we defined A as the union of all the champion lists of the query terms and we calculate the cosine similarity only for documents in A

Potential problems: the size of C is critical and application dependent. It may be vary for different terms, for example can be set to higher values for rarer terms

Static quality scores

Many systems have access to a static measure of quality for document that contribute to the score

Example of static measures are user query preferences or feedback, re-tweet, "like"

These static values can be used to sort the posting list and select only the top-k postings for matching (like in the champion list approach)

Impact ordering

Another idea based on sorting the posting lists is based on sorting by decreasing order of tf with respect to the query term at hand

Cluster pruning

Here, we perform a preprocessing step in which documents are clustered as follows:

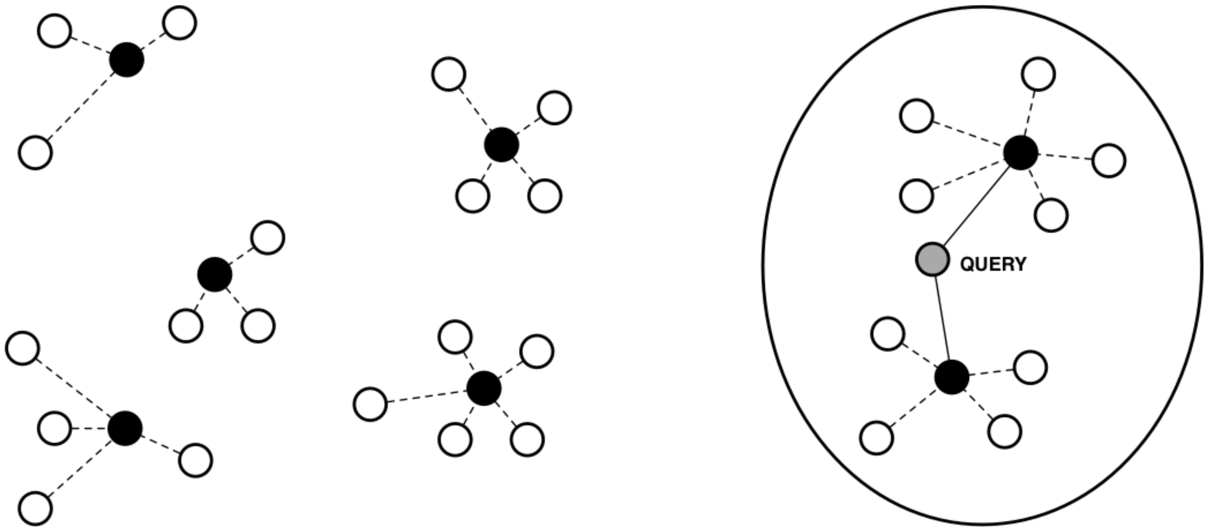
- Randomly choose $\sqrt{N}$ documents from the corpus as **leaders**
- For each non-leader document (called **follower**, compute the nearest leader

Given a query q compute similarity with respect to the leaders only

For each leader in the top-k results, compute similarity also for its followers

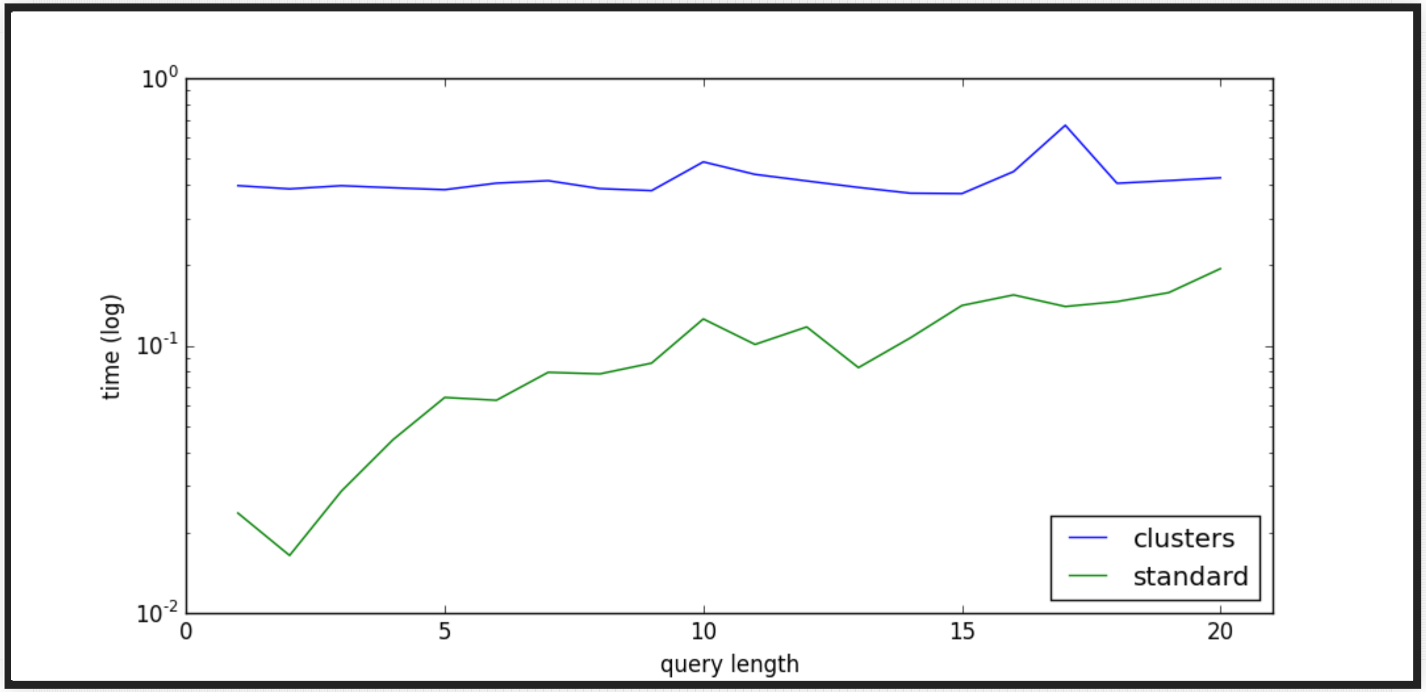
The random choice of leaders may the substituted with other criteria of leader selection

Example



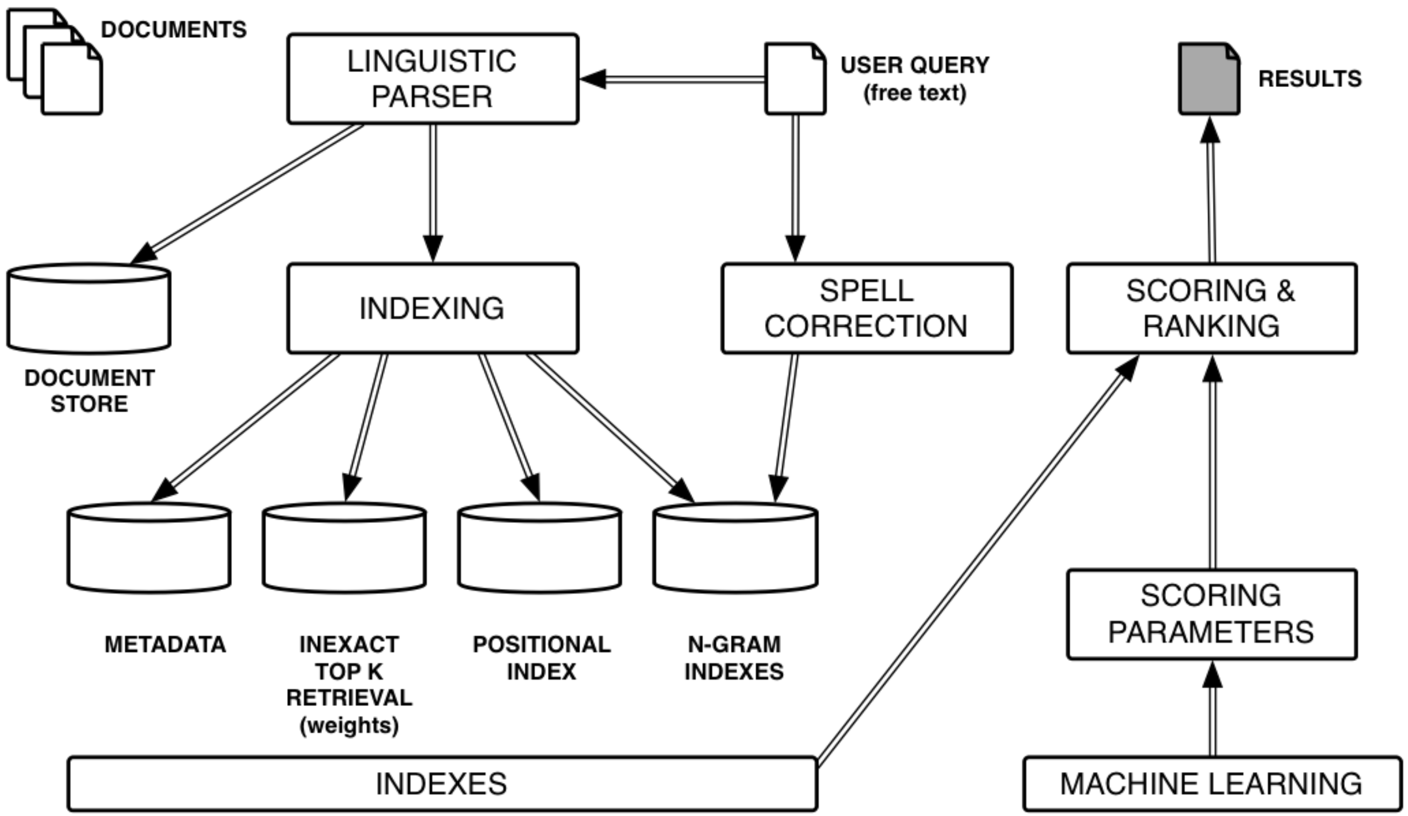
We executed 7 similarity evaluation operations (with the leaders) and 9 with the followers of the selected leaders

EXAMPLE



10^0
10^{-1}
10^{-2}
time (log)
0
5
10
15
20
query length
clusters
standard

ARCHITECTURE OF A COMPLETE SYSTEM



See also Figure 7.5 on the textbook

RECAP

Topic	Chapters
Term weighting	6
Foundations of the vector space model	6
Heuristic approaches for fast query processing	7