Week 1 - Retrieval Models

The Basic Boolean Retrieval

The simplest **Exact Match** model:

- Retrieve documents if they satisfy a Boolean expression
- Query specifies precise relevance criteria
- Documents returned in no particular order

Representing a **document** with a bag of words and a **query** with a boolean expression.

	a	Aachen	abandon	abate	...	zygote
Doc_1	1	1	0	0	...	1
Doc_2	1	0	1	1	...	0
Doc_3	1	0	0	1		0
					...	
Doc_N	1	1	1	0	...	0
Query	**0**	**1**	**1**	**0**	**...**	**0**

Works well if you know **exactly what you want**:

- Structured queries
- Simple to program
- Complete expressiveness
- Computationally efficient

Disadvantages:

- Difficult to balance precision and recall
- Unordered output

Extensions to Boolean retrieval:

- **proximity operators**: impose constraints on relative position of query-terms
- **field restriction**: impose constraints on location of query-terms, e.g. Title, Abstract
- **wild-card operators**: impose constraints on matching query-terms with index-terms

Proximity Operators

Ordered window: term A must appear no more than N terms before Term B

$$\text{A OW/N B}$$

For example:

Paris OW/2 Climate

Paris climate change accord

Paris hosts climate change

Climate of Paris

Paris climate in summer

Unordered window: term A must appear no more than N terms from Term B

$$A\ UW/N\ B$$

Phrase: term A must appear immediately before Term B

$$"A\ B"$$

Ranked Boolean Retrieval

Model and operators are the same as for Boolean retrieval, the only difference is that matched documents are **ranked by frequency of query terms**:

- Document term weights: how often a term occurs in a document – may be normalized
- AND weight: Minimum of argument weights
- OR weight: Maximum of argument weights
- and, sum of all argument weights

For example:

- Query is “brown” AND “cat”
- Document1 contains 3 occurrences of “brown” and 5 of “cat”
  - Score = min(3,5) = 3
- Document2 contains 4 occurrences of “brown” and 5 of “cat”
  - Score = min(4,5) = 4
- Document2 is more relevant

Vector Space Representation

Relevance is based on how close a document (in vector space) is to a query (in vector space).

Documents as vectors:

- we have a |V|-dimensional vector space, where |V| is the vocabulary size
- Terms are axes of the space (representing the frequencies)
- 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 (inverted index exploits this)

Query as vectors:

- Rank documents according to their proximity to the query in this space
- proximity = similarity of vectors
- proximity ≈ inverse of distance
- Recall: We do this because we want to get away from the “either-in-or-out” Boolean model.
- The intent is to rank more relevant documents higher than less relevant documents

Query-Document Matching Scores

Co-ordination Level Matching: counts the **number of terms in common** between a query and a document.

$$score(q, d) = |g \cap d| = \sum_{t \in (g \cap d)} 1$$

This is equivalent to the **inner product** between the document and query vectors:

$$score(q, d) = \sum_{i=1}^{V} d_i \times q_i$$

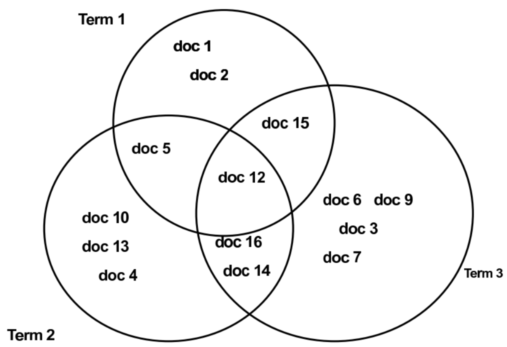
This is also equivalent to:

$$score(q, d) = V - \text{hamming distance}$$

where **hamming distance** measures the number of positions that differ between two vectors.

- Suppose we query “term#1 term#2 term#3”

Co-ordination level	*docs*
3	doc 12
2	doc 15 doc15 doc16 doc 14
1	the rest



Disadvantages:

- documents with more words will be more likely to be relevant
- does not consider the frequencies of terms appearing in the documents

Term Frequency (TF)

Term Frequency (TF) is the number of times the term occurs in a document.

Instead of using binary vectors, we can use **frequecies of occurrence** as weights:

$$score(q, d) = \sum_{t \in |q \cap d|} TF_{t,d}$$

where $TF_{t,d}$ is the frequency of occurance of a term t in document d.

This is equivalent to:

$$score(q, d) = \sum_{i=1}^{V} d_t \times q_t$$

The **term frequency distribution**:

- obeys the **Zipf's law** such that states that the i^{th} most frequent term has frequency proportional to $\frac{i}{i}$
- about half of all vocabulary terms occur only once in the collection

However, in term frequency score, all terms are treated equally and therefore **common words** have more weight. For example, the retrieved documents might not contain the word "fudgel" but instead contains a lot of "what", "does" and "mean" when the query is **"What does fudgel mean"**.

Inverse Document Frequency (IDF)

Query terms are different in their ability to **discriminate documents**. A query term is **not a good discriminator** if it occurs in many documents (e.g. c**ommon words** like "is" and "do"). Therefore, we should give it **less weight** than the one occurring in few documents.

Inverse Doucment Frequency (IDF) is a measure of term specificity:

$$IDF_t = \log_{10}(\frac{N}{n_t})$$

where the N is the **number of documents** in collection and n_t is the number of documents in which **term** t **appears**.

IDF example, suppose *N* = 1 million

term	n_t	IDF_t
calpurnia	1	6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0

IDF affects the ranking of queries with **at least two terms**. For example, the query **“essex council”**, IDF weighting makes occurrences of “essex” count for **much more** in the final document ranking than occurrences of “council”.

tf.idf weighting

$$tf_t \times idf_t$$

greater when the term is **frequent** in in the document

greater when the term is **rare** in the collection (does not appear in many documents)

Therefore, the **similarity score** with TF-IDF weighting is given as:

$$score(q, d) = \sum_{t \in (q\ capd)} tf_t \times idf_t = \sum_{t=1}^{V} d_t \times q_t$$

where d_t is the **term frequencies** of documents and q_t is the **inverse document frequencies** of query terms.

Problems with Inner Product of Vectors

There exists a very **strong bias** such that **longer documents** ar elikley to score higher (because of better word coverage and frequency).

Therefore, we need to come up with a **distance** measure between two vectors that does not affect by the **length** of the documents.

Euclidean distance is a bad idea, but **cosine similarity** works.

Cosine Similarity

The **numerator** is the inner product and the **denominator** is the product of the two vector-lengths.

The consine similarity score ranges from **0 to 1** and it measures the **cosine of the angle** between two vectors:

Inner product

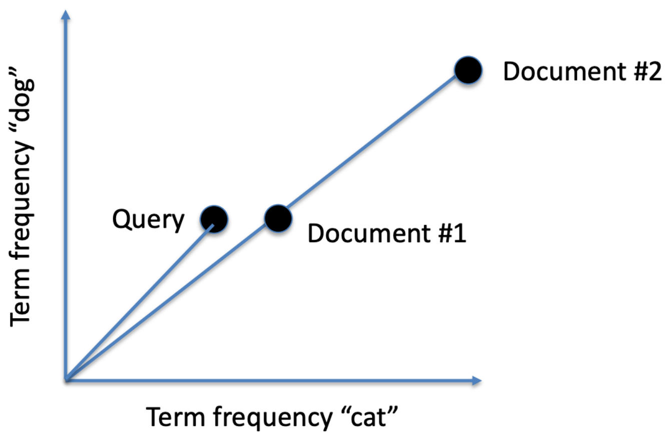
$$\frac{\Sigma_{i=1}^{V} x_i \times y_i}{\sqrt{\Sigma_{i=1}^{V} x_i^2} \times \sqrt{\Sigma_{i=1}^{V} y_i^2}}$$

Length of x Length of y

In terms of the **query vector** and the **document vector**:

$$\frac{q.d}{|q||d|} = \frac{\sum_{t \in (q \cap d)} q_t \times d_t}{\sqrt{\sum_{t=1}^{V} q_t^2} \sqrt{\sum_{t=1}^{V} d_t^2}}$$

It can be considered as **penalizing** the similarity score by the **vector length**. That is, the similarity score will be **suppressed** if the vectors are long.



Cosine similarity the same between Query and Doc#1, and Query and Doc#2