

# Information retrieval

Francesco Tomaselli

April 13, 2021

## Contents

<b>1</b>	<b>Vector space model</b>	<b>2</b>
1.1	Tokenization . . . . .	2
1.2	Normalization . . . . .	3
1.3	Term Weighting . . . . .	3
<b>2</b>	<b>Evaluation on retrieval systems</b>	<b>4</b>
2.1	The notion of quality . . . . .	4
2.2	Quality measures . . . . .	5
2.3	Evaluation of ranking systems . . . . .	6
<b>3</b>	<b>Relevance feedback and query expansion</b>	<b>7</b>
3.1	Local methods . . . . .	8
3.2	Global Methods . . . . .	8
3.2.1	How to add information? . . . . .	9
3.2.2	Spelling correction . . . . .	10
<b>4</b>	<b>Probabilistic Information Retrieval</b>	<b>10</b>
4.1	Review of probability theory . . . . .	11
4.2	Binary independence model . . . . .	12
4.3	Non binary models . . . . .	14
4.4	Probabilistic language modelling . . . . .	14
4.5	Other models . . . . .	16
<b>5</b>	<b>Unsupervised text classification</b>	<b>16</b>
5.1	Clustering . . . . .	16
5.1.1	Support to document retrieval . . . . .	17
5.1.2	Evaluation . . . . .	17
5.2	K-means . . . . .	17
5.3	Model-based clustering . . . . .	18
5.3.1	Expectation Maximization algorithm . . . . .	18
5.4	Affinity propagation clustering . . . . .	19
5.5	Hierarchical clustering . . . . .	20
<b>6</b>	<b>Topic modeling</b>	<b>21</b>
6.1	Linear algebra recap . . . . .	21
6.2	Latent Semantic Indexing . . . . .	22
6.3	Latent Dirichlet Allocation . . . . .	23

# 1 Vector space model

**Text transformation** The first step to process queries on text is to transform it into a model that can be computed by a machine.

Such a model should be suitable for any kind of text, should be able to capture the semantic of words and then should support the notion of *text similarity*.

The main idea of a vector space model is to map documents in a vector, so they appear as points in a certain space.

For instance, we can collect some words and index them, then build a vector for each document

$$d_k = [x_0, \dots, x_n]$$

where the value  $x_i$  stores the occurrences of the word indexed with  $i$  in that document.

**Similarity** This model somehow implements the notion of text similarity, represented by some sort of distance function between two documents in space, perhaps the *Euclidean distance* or the difference between the angle of the two vectors. There are a lot of distance measures, and they vary in applications. Some of them assume normalization in the document vectors.

**Document-Term matrix** If we arrange document vectors in a vector we obtain the *document-term matrix*, or the *term-document matrix* if we take the opposite. The number of dimensions in the space is equal to the number of words in the vocabulary.

That can be a problem without some sort of lemmatisation and stemming, as sparsity and dimension of the matrix increases.

**Vector length problem** By considering the length of the vector in a similarity measure, we are introducing some problems, let's just consider a text and its abstract, the Euclidean distance would be huge, as most of the words do not appear that often.

To solve the problem we can consider the *Cosine similarity* or normalize vector lengths.

The main phases in the generation of a vector space models are: tokenization, normalization, weight, and indexing.

## 1.1 Tokenization

The goal is to split the text in words, for instance we could split the document at the spaces.

In practice this can be obtained with *Regex*, *Corpus* based and *Machine Learning* based methods.

## 1.2 Normalization

The idea is to normalize each token, that could mean writing verbs in the normal form, or plurals at the singular.

**Stemming** One form of normalization is stemming, that simply truncate words to obtain singular or infinite forms. The problems here could be creation of meaningless words and also close words in meaning could be completely different. An example is the *Porter* stemming algorithm.

**Lemmatisation** Another way to normalize is to lemmatise, It's similar to stemming, but the words are replaced with the dictionary root, this also has a problem, indeed sometime mapping of same words to different lemmas can happen. A way to lemmatise is to use a *Dictionary based* lemmatisation, or something more complicated such as *Wordnet*.

**Wordnet** Wordnet is a large lexical database, verbs, nouns, adjectives and adverbs are grouped into cognitive synonyms, called synsets, each expressing a different concept. Synsets are connected to each other by conceptual relations and they are also connected to a lemma.

## 1.3 Term Weighting

Weighting the words is a crucial point in text processing, an easy way could be counting the frequency of the terms in the document.

**Term frequency** An example of weighing is the term frequency, which counts the frequency of a term given a document:

- *Boolean*: so a 0 or 1 if the word is present or not
- *Natural*:  $tf_{t,d}$  = count of a specific word
- *Log*:  $1 + \log(tf_{t,d})$
- *Augmented*:  $0.5 + \frac{0.5tf_{t,d}}{\max_{t'} tf_{t',d}}$
- *Log average*:  $0.5 + \frac{0.5 \log(tf_{t,d})}{1 + \log(\max_{t'} tf_{t',d})}$
- *Max tf norm*:  $k + (1 - k) \frac{tf_{t,d}}{tf_{\max}(d)}$

**Stop words problem** A problem in this model is the excessive presence of stop words and punctuations. To compensate, we can remove them in the normalization phase.

But sometimes the stop words can be important, for instance with phrasal verbs. A solution can be to count the number of documents that contains that given word.

**Inverse document frequency** The idea is to count the number of documents that contains a given words, and to prefer less frequent words:

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

There are many variants:

- *Smooth*:  $\log(\frac{N}{1+df_t}) + 1$
- *Max*:  $\log(\frac{\max_{t' \in d} df_{t'}}{1+df_t}) + 1$
- *Probabilistic*:  $\log \frac{N-df_t}{df_t}$

**TfIdf** By multiplying term frequency and inverse term frequency we obtain this metric.

$$TfIdf(t, d) = tf_{t,d} \cdot idf_t$$

Regarding the value of terms:

- the terms with a high *TfIdf* usually appears in a small number of documents
- the metric is low when a term appears a few times in a document or when it appears in many documents

## 2 Evaluation on retrieval systems

The goal of evaluation is to assess the quality of the results obtained by an IR system. There's the need of knowing a *ground truth*, so an annotated corpus where for each task we know what documents are relevant. The annotations could be created manually or derived from the data, if it contains annotations.

### 2.1 The notion of quality

Given a corpus  $C$  and a query  $q$ , the task is to find a set of documents  $A_{q,C}$  that match  $q$ , but after retrieving such documents, there's the need to estimate the quality of results.

**Precision** To formalise the quality of the retrieved documents  $A_{q,C}$  we could count how many of these documents are relevant to  $q$ , this is the notion of precision:

$$Prec = \frac{\text{relevant retrieved}}{\text{retrieved}}$$

Note that in order to know if a document is relevant or not we need the ground truth or a user feedback.

This measure does not suffice, as for instance, if we retrieve only one correct document we would have maximum precision.

**Recall** The precision measure does not take in consideration how many relevant documents are there, that is crucial to assess quality of a query result. So we can consider another measure, called Recall:

$$Rec = \frac{\text{relevant retrieved}}{\text{relevant}}$$

**Information need** Given the two quality measures, should we aim at better precision or more recall? Trivially both, but it actually depends on the information need.

In some cases a really high recall is not necessary, while other times a lower precision could be accepted while not missing anything relevant.

**F1 Score** To take into consideration both precision and recall, we could take a weighted mean, so a tradeoff between the two

$$F1 = \frac{2 \cdot Prec \cdot Rec}{Prec + Rec}$$

**Baseline system** To assess the quality of an IR system, we need a baseline system, something trivial such as tossing a coin to decide whereas a document is relevant or not. Given its quality measure, we can infer the quality of our, hopefully, more complex system, in fact, quality measures can't be seen as absolute values, there's always the need to compare.

## 2.2 Quality measures

To formally define the notions introduced in the previous section, we need to take into consideration a more detail measure for errors.

**Confusion Matrix** Given a query  $q$ , a ground truth  $E_q$  of relevant documents with respect to  $q$ , and a set of retrieved documents  $A_q$ , we define this matrix:

	$d \in E_q$	$d \notin E_q$
$d \in A_q$	True positive	False positive
$d \notin A_q$	False Negative	True Negative

So now we can redefine the *precision*, *recall* and *F1* measures as follows

$$Prec = \frac{TP}{TP + FP} \quad Rec = \frac{TP}{TP + FN} \quad F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

The actual confusion matrix is obtained by counting the documents given the retrieved ones and the ground truth.

The matrix can be useful to estimate the system parameters, for instance, if the number of retrieved documents is set to a certain value  $k$  and the value of true positives is really high, probably a smaller  $k$  would do the job, while if a lot positives are left out, i.e, the matrix has a high number of false negatives, maybe an higher  $k$  would make the system better.

**Other measures** There are a lot of measures to estimate the quality of a system, such as *specificity*, *negative predictive value*, *miss-rate*, *fall-out* and *accuracy*. Finding the right measure is really important, for instance, using accuracy with an unbalanced dataset is not better than precision, recall and F1.

## 2.3 Evaluation of ranking systems

We talked about boolean retrieval systems and we took into consideration if a document is retrieved or not.

In a ranking scenario, we want to give importance not only to precision and recall, but also to the rank assigned by the system.

**Setting a threshold** One way to go is to set a specific threshold and compute the precision and recall of those top documents. This method does not take into consideration the ranking of the documents.

**Precision at K** If we take the first  $K$  retrieved documents, ordered by rank, we can estimate the quality of the rank by computing the precision for the top  $K$  documents. By iterating the threshold we can better estimate the performances, opposed to setting a unique threshold.

To decide what thresholds to use we could use the distribution of the system ranking.

**Discounted cumulative gain** This approach takes into consideration the ranking and discount the gain given by a document with respect to its position in it.

$$DCG = \sum_{i=1}^n \frac{R_j}{\log(i+1)}$$

where  $R_j$  is the rank assigned to a document.

**Precision vs Recall curve** Another approach is to compute precision and recall measures at each point in the ranking. If we take the measurements and plot them, we find a correlation between precision and recall and if we compute the integral of that function, we obtain the average precision.

To have a smoother curve, it's possible to interpolate the precision, so for each point in the recall axis, we take the maximum precision of consecutive points.

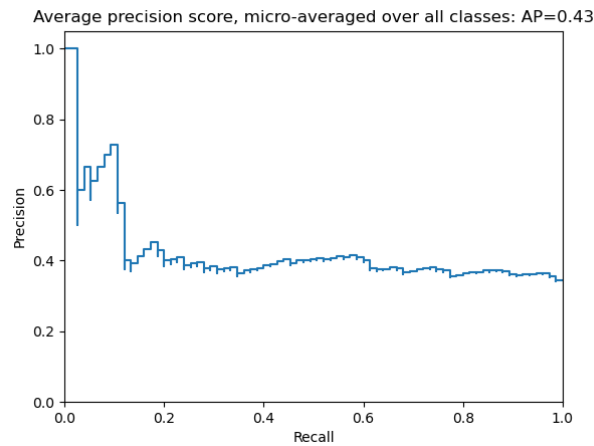


Figure 1: Example of a non-interpolated precision and recall curve

### 3 Relevance feedback and query expansion

The idea behind this section is the possibility to use some sort of feedbacks to tweak queries before passing them to the information retrieval system.

**Relevance feedback** It's any feedback we can collect about correctness of a certain query. For instance a user feedback or a indirect feedback about usefulness of an answer, i.e. the user clicked on a particular website

**Query expansion** Transform user queries exploiting the relevance feedback to improve system results over that query.

When expanding a query there are two possible methods, local and global methods, depending on whereas the expansion is applied to a single query or to all queries to the system:

- *Local methods*: related to a particular query, they are based on relevance and indirect feedback of that particular query
- *Global methods*: methods applied to all queries, those can be expansion with a thesaurus, or spelling correction

### 3.1 Local methods

**Vector shifting** Given the sets  $R$  and  $N$  of relevant and non relevant results respectively, the task is to find a query vector  $\vec{q}_o$  that maximizes similarity with  $R$  and minimizes the one with  $N$ :

$$\vec{q}_o = \operatorname{argmax}_{\vec{q}} [\operatorname{sim}(\vec{q}, R) - \operatorname{sim}(\vec{q}, N)]$$

Given a similarity function, for instance the cosine similarity, the equation becomes

$$\vec{q}_o = \frac{1}{|R|} \sum_{\vec{d}_i \in R} \vec{d}_i - \frac{1}{|N|} \sum_{\vec{d}_j \in N} \vec{d}_j$$

where the two elements are the centroids of  $R$  and  $N$ .

**Rocchio algorithm** This approach is a generalization of the main idea introduced in the section, it proposes a new vector derived by making the query vector closer to the centroid of relevant documents.

$$\vec{q}_m = \alpha \vec{q} + \beta \frac{1}{|R|} \sum_{\vec{d}_i \in R} \vec{d}_i - \gamma \frac{1}{|N|} \sum_{\vec{d}_j \in N} \vec{d}_j$$

where  $\alpha$ ,  $\beta$  and  $\gamma$  can balance the role of each component.

One of the problems of this approach is the difficulty in estimating a document relevance, also, the sets  $R$  and  $N$  could be somehow merged.

**Pseudo-relevance feedback** Instead of collecting real feedback for a query, we assume that the top  $k$  results are correct, we then use them as a feedback for the Rocchio algorithm.

To motivate this choice we can look at the precision and recall curve, indeed a small set of results usually lead to high precision, thus we can move the system in the direction of the top results.

**Indirect relevance feedback** In this case the feedback is obtained by looking at user behavior, for instance we could count the number of visits to the results to measure the popularity of documents.

### 3.2 Global Methods

**Query expansion** The idea here is to expand the query by adding new information, this could mean adding terminology or shifting the query vector as before.



### 3.2.1 How to add information?

To add information to a query we can perform *syntactic analysis* of the original query, using *dictionaries* to add new terms related to the query, or use the *corpus of reference documents* to add terminology.

**Dictionaries and knowledge bases** The process to enrich a query when having a knowledge base, that could be a dictionary an ontology or whatever, is to:

1. *Lookup* the original query on the external knowledge base
2. Extract *candidate terminology*, so terms that could be added to the query and are somehow related to the one in the original string
3. Use a *word sense disambiguation* tool to split relevant and non relevant terms, and add the relevant ones to the original query

**Wordnet** As introduced in section 1.2 on page 3, Wordnet can be used to extract lemmas but also to find hypernyms and hyponyms of a given word. For instance, if a query presents the word *play*, with the meaning of a *dramatic composition* we could add the latter to the original query.

Another use case could be searching for something like *President Lincoln*. In this case we check for the definition of the two terms, and find out that there's a matching in some possible definitions, indeed the first one could be related to the president of the US, and also the second, thus we can infer that the query is related to US presidency and enrich the query with this information.

**Statistical terminology expansion** Given the set of relevant and non relevant documents  $R$  and  $N$  we can select relevant terminology by using the *pointwise Kullback-Leibler divergence*. We compute the probability of a word to be in  $R$  and in  $N$

$$p(w) = \frac{\text{count}(w, R)}{\sum_{w_i \in R} \text{count}(w_i, R)}, \quad q(w) = \frac{\text{count}(w, N)}{\sum_{w_i \in N} \text{count}(w_i, N)}$$

and then compare them

$$\delta_w(p || q) = p(w) \log \frac{p(w)}{q(w)}$$

if this quantity is close to zero a term is not useful to separate relevant and non relevant documents. On the other end, words with a positive  $\delta$  can be used to expand the query.

### 3.2.2 Spelling correction

There are main phases in spelling correction, the first one is to select the correct versions of a certain query in a set of candidates and then choosing the best one, i.e. the most common or the nearest one.

To detect a *spelling error* we could see few query results, or maybe search for the query tokens in a vocabulary.

**String similarity** A measure that computes the distance of two strings, one could be the *Levenshtein distance*, also known as the edit distance. It can be solved with dynamic programming implementing the following recursive definition:

$$DP[i][j] = \begin{cases} j & \text{if } i = 0 \\ i & \text{if } j = 0 \\ DP[i-1][j-1] & \text{if } s[i] = r[j] \\ \min(DP[i-1][j], DP[i][j-1], DP[i-1][j-1]) + 1 & \text{otherwise} \end{cases}$$

**Phonetic correction** To minimize errors due to phonetic misspelling the idea is to use a *phonetic hash*, so that similar sounding words have the same value. These algorithm are called *Soundex* and are language-dependent.

**K-gram classification and matching** The idea is to consider for each word the subsequences of  $K$  characters.

An example, for the word *PLAY*, is to first add a start and end symbol,  $\#s$  and  $\#e$  respectively and then computing the subsequences, counting how many times they appear in the word.

This creates a vector representation for the words, and to find similar words we can use cosine similarity like we did in section 1 on page 2.

**Sequence-to-sequence learning** This technique use deep learning to match a sequence into another one. The idea is to train a model to map misspelled words into they correct counterpart.

## 4 Probabilistic Information Retrieval

The complexity of text analysis is such that we can't have a notion of correctness of an answer to a query, we therefore switch idea and change *true* to *probable*.

## 4.1 Review of probability theory

**Random variable** A variable  $A$  represents an events, so a subset of the space of possible outcomes. Another representation of  $A$  is through a random variable, that maps outcomes to real numbers.

**Joint and conditional probabilities** We denote  $P(A|B)$  the probability of  $A$  occurring given the occurrence of  $B$ .

**Chain rule** The following twos hold

$$P(A, B) = P(A \cap B) = P(A|B)P(B) = P(B|A)P(A)$$

$$P(\neg A, B) = P(\neg A|B)P(B)$$

**Partition rule** If an event  $B$  can be divided into an exhaustive set of disjoint sub-cases, the probability of  $B$  is the sum of the sub-cases

$$P(B) = P(A, B) + P(\neg A, B)$$

**Bayes rule** The following rule holds

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

This rule is a powerful tool to invert the conditional probability. The first step is to estimate the likelihood of the event  $A$ , thus  $P(A)$ , then we can derive a posterior probability  $P(A|B)$  after having seen the event  $B$ , based on the likelihood of  $B$  occurring when  $A$  does or not hold.

The left member of the equation  $P(A|B)$  is what we are trying to estimate, this could be for instance *probability of a document to be relevant given that it contains a word  $X$* .

$P(B|A)$  is on the other end the probability of observing the new evidence given the initial hypothesis. This could mean finding the *probability of observing the word  $X$  given that a document is relevant*.

$P(A)$  is called the prior. For instance *probability of any document to be relevant*.

$P(B)$  is the marginal likelihood, for instance *probability of observing the word  $X$  in any document*.

**Odds of an event** It is often useful to talk about the odds of an event

$$O(A) = \frac{P(A)}{P(\neg A)} = \frac{P(A)}{1 - P(A)}$$

## 4.2 Binary independence model

In information retrieval we represent the relevant of a document given an information need as a random variable  $R$ . Given a document  $d$  and a query  $q$  we want to estimate the probability

$$P(R = 1|q, d)$$

Given a dictionary of terms  $T = \langle t_1, \dots, t_n \rangle$  we model each document as a binary vector  $\vec{x} = (x_1, \dots, x_n)$  where  $x_i$  is 1 if the  $i$ th term is in the document.

The goal is to calculate

$$O(R = 1|q, \vec{x}) = \frac{P(R = 1|q, \vec{x})}{P(R = 0|q, \vec{x})}$$

**Simplifying the goal function** Applying the Bayes rule leads to

$$\begin{aligned} O(R = 1|q, \vec{x}) &= \frac{P(R = 1|q, \vec{x})}{P(R = 0|q, \vec{x})} = \frac{\frac{P(\vec{x}|R=1,q)P(R=1|q)}{P(q,\vec{x})}}{\frac{P(\vec{x}|R=0,q)P(R=0|q)}{P(q,\vec{x})}} \\ &= \frac{P(\vec{x}|R = 1, q)}{P(\vec{x}|R = 0, q)} \cdot \frac{P(R = 1|q)}{P(R = 0|q)} \end{aligned}$$

The left part of the result represent the odds of having that document given that it is relevant and given that query. On the right part we have the probability of the query producing relevant documents or not.

Assuming that the terms are independent, note that this is a strong assumptions, we can write

$$O(R = 1|q, \vec{x}) = \frac{P(R = 1|q)}{P(R = 0|q)} \prod_{i=1}^n \frac{P(x_i|R = 1, q)}{P(x_i|R = 0, q)}$$

This product can be split according to the presence of terms in the document

$$O(R = 1|q, \vec{x}) = \frac{P(R = 1|q)}{P(R = 0|q)} \prod_{i=1}^n \frac{P(x_i = 1|R = 1, q)}{P(x_i = 1|R = 0, q)} \prod_{i=1}^n \frac{P(x_i = 0|R = 1, q)}{P(x_i = 0|R = 0, q)}$$

To simplify the formula we introduce this values

$$p_i = P(x_i = 1|R = 1, q), \quad q_i = P(x_i = 1|R = 0, q)$$

	Document	$R = 1$	$R = 0$
We observe the term	$x_i = 1$	$p_i$	$q_i$
We do not observe the term	$x_i = 0$	$1 - p_i$	$1 - q_i$

Thus the previous equation becomes, where the first product iterates over the words that appear in the document and the query, and the second product goes

for words that do not appear in the document, but do on the queries.

$$O(R = 1|q, \vec{x}) = \frac{P(R = 1|q)}{P(R = 0|q)} \prod_{x=1, q=1} \frac{p_i}{q_i} \prod_{x=0, q=1} \frac{1-p_i}{1-q_i}$$

This assumes that  $p_i = q_i$  for words that don't appear in a query, the assumption is motivated by the fact that a document is relevant or not given a query, not by itself.

We now extend the right product to cover all the query terms, formally

$$O(R = 1|q, \vec{x}) = \frac{P(R = 1|q)}{P(R = 0|q)} \prod_{x=1, q=1} \frac{\frac{p_i}{q_i}}{\frac{1-p_i}{1-q_i}} \prod_{q=1} \frac{1-p_i}{1-q_i}$$

We note that the quantity  $\frac{P(R=1|q)}{P(R=0|q)}$  represents the odds of the query and does not depend on the document. Also, the quantity  $\prod_{q_i=1} \frac{1-p_i}{1-q_i}$  depends on the terms that are in the query, it's so query dependent but it does not change for different documents. This leads to a simplification of the previous formula, with some rewriting

$$O(R = 1|q, \vec{x}) \approx \prod_{x_i=1, q_i=1} \frac{p_i(1-q_i)}{q_i(1-p_i)}$$

**Retrieval Status Value** After the last simplification we can calculate the Retrieval Status Value for a document given a query as the quantity

$$RSV_{q,d} = \log \prod_{x_i=1, q_i=1} \frac{p_i(1-q_i)}{q_i(1-p_i)} = \sum_{x_i=1, q_i=1} \log \frac{p_i(1-q_i)}{q_i(1-p_i)}$$

**Estimating probabilities** We now just need to estimate  $p_i$  and  $q_i$  and we are ready to go. If a dataset has informations about relevant documents given a query, we can easily estimate them, given  $R(t)$  and  $N(t)$  occurrences of term  $t$  in relevant and non relevant documents

$$p_i = \frac{R(t_i) + 0.5}{R + 1.0}, \quad q_i = \frac{N(t_i) + 0.5}{N + 1.0}$$

Of course we don't always have the relevant and non relevant documents for queries, so we need some assumptions:

- $p_i = q_i$  if  $t_i \notin q$ , as discussed above
- $p_i = 0.5$  if  $t_i \in q$ , we have the same probability of observing or not a term in a randomly-picked relevant document
- $q_i \approx \frac{|\{d|t_i \in d\}|}{|D|}$ , the set of non-relevant documents is approximated by the whole collection of documents  $D$

By applying the previous assumptions we obtain

$$\begin{aligned}
RSV_{q,d} &= \sum_{x_i=1, q_i=1} \log \frac{p_i(1-q_i)}{q_i(1-p_i)} \\
&= \sum_{x_i=1, q_i=1} \log \frac{(1-q_i)}{q_i} \\
&= \sum_{x_i=1, q_i=1} \log \frac{N - N_i + 0.5}{N_i + 0.5}
\end{aligned}$$

By having  $n$  and  $N$  as the number of docs with  $t$  and the total number of docs, respectively

$$\log \frac{(1-q_i)}{q_i} \approx \log \frac{N-n}{n} \approx \log \frac{N-n}{n} = IDF$$

This means that in practice, we are intersecting a document and a query terms, summing the *IDF* of the words in the intersection, of course, by considering the initial assumptions.

### 4.3 Non binary models

The non binary models takes into consideration the frequency of the terms and not simply a binary vector. They remove the binary assumption of the model presented in 4.2 on page 12.

**Okapi BM25** In this non binary model, the RSV is estimated as follows

$$RSV_{q,v} = \sum_{t \in q} \log \frac{N}{df_t} \cdot \frac{(k+1)tf_{t,d}}{k((1-b) + b(\frac{L_d}{L_{avg}})) + tf_{t,d}}$$

$k$  regulates the term frequency, while  $b$  scales the importance of the document length.

### 4.4 Probabilistic language modelling

The goal is now to remove the independence assumption made in the model at 4.2 on page 12.

In order to accomplish this, we need a model that can compute the probability of a sentence, i.e. a query, where the order of the words is important:

$$P(S) = P(w_1, \dots, w_n)$$

Another request is to have a generative model for sentences, so a model that estimates the probability of a word, given the previous terms in the sequence:

$$P(w_k | w_{k-1}, \dots, w_1)$$

The idea is to build a model on top of each document and calculate the probability  $P(q|M_d)$  that a query  $q$  has been generated by the model  $M_d$ .

**Computing a sentence probability** Given a sentence, for instance *Mario plays a lot with Maria*, we need to compute the joint probability:

$$P(\text{Mario, plays, a, lot, with, Maria})$$

By applying the chain rule we can calculate the probability of the sequence

$$P(x_1, x_2, \dots, x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1, x_2) \cdots = \prod_i^n P(w_i|w_1, \dots, w_{i-1})$$

Calculating the quantity on real world documents could be impossible, as we might not be able to count long sequences of words to calculate the probability. An approximation to the sequence probability is given by restricting the sequence length:

$$P(w_1, \dots, w_n) \approx \prod_i^n P(w_i|w_{i-k}, \dots, w_{i-1})$$

$k$  is a parameter, it should be small if the dataset is really small, or higher if there is enough data. For instance, if we take  $k = 2$  we are estimating the sequence by bi-grams

$$P(w_1, \dots, w_n) \approx \prod_i^n P(w_i|w_{i-1})$$

Note that to compute the probability of a bi-gram, we just count occurrences of a given bi-gram over all the possible ones.

**Query likelihood model** After building a model  $M_d$  for each documents, following the idea presented on the previous paragraph, we can estimate relevance of documents given a query.

So the probability that a query was generated by a document becomes:

$$P(q|M_d) = K_q \prod_i P(w_i|M_d)^{tf_{w_i,d}} \quad K_q = \frac{Len(K_q)!}{tf_{t_1,q}! \cdots tf_{t_N,q}!}$$

The  $K_q$  is query dependent, so we can ignore it, it would take into consideration the query permutations.

We can also estimate the probability with this quantity

$$P(q|M_d) \approx \prod_i \frac{tf_{t_i,d}}{L_d}$$

Note that  $tf_{t_i,d}$  can be changed to the probability of the k-grams computed at the previous paragraph, it depends on how we choose to estimate the sentence probability.

**Smoothing** Note that all zero and non frequent term probabilities are a problem, for instance a long sequence could be interrupted by only one bi-gram not appearing in the document, the solution is to smooth the numbers, for instance with the *Laplace smoothing* that adds one to each bi-gram, the *Bayesian smoothing* or the *Linear interpolation* that takes into consideration the global document model.

## 4.5 Other models

**Kullback-Leibler divergence model** In this model the relevance term is computed as the quantity

$$R(d, q) = \sum_i P(w_i | M_q) \log \frac{P(w_i | M_q)}{P(w_i | M_d)}$$

where  $M_q$  is the model computed on the query.

**Translation model** The idea here is to use a generative model taking into consideration the possible translation of a query term.

## 5 Unsupervised text classification

**Text classification** The problem of associating texts, so documents, to classes, represented by labels. Classes may be a partition of the document space, or they can overlap. The approaches to solve the problem are basically two, pre-trained models, so supervised, or based only on documents, so unsupervised.

Depending on the number of classes and how they cover the document space, we define four tasks

	2 classes	Many classes
Disjoint classes	Binary classification	Multi-class classification
Overlapping classes	Soft binary classification	Multi-label classification

Note that ranking can be seen as soft binary classification.

### 5.1 Clustering

**Goal** Clustering is the problem of grouping objects in clusters, in particular given a distance metric, we want to minimize the distance within a cluster and maximize the one between different clusters.

**Parameters** Choosing the right number of clusters is pretty difficult, a high number leads to consistency inside clusters, but poor aggregation, while a low number increases aggregation but reduce quality of the single clusters.



**Types of clustering** There are many approaches to clustering, one could for instance consider flat or hierarchical clustering, in the first one there is no structure, where in the second clusters might contain other clusters. There is also a difference in cluster assignment, one could assign in an hard or soft way.

#### 5.1.1 Support to document retrieval

**Terminological analysis** After computing a clustering of documents we could analyze terminology of documents of the same cluster to extract relevant terms, this information can then be used for query expansion.

**Cluster pruning** By selecting representative documents for clusters, we can compare a query only to those documents. After finding a match all the documents in the cluster are returned.

#### 5.1.2 Evaluation

Given  $\Omega = \{\omega_1, \dots, \omega_n\}$  the set of clusters and  $\mathbb{C} = \{c_1, \dots, c_n\}$  the set of classes, we can evaluate the clustering by focusing on pairs of documents.

Let's consider the pair  $(x_i, x_j)$ , we ask ourselves if

$$\exists \omega_k : (x_i, x_j) \in \omega_k \quad \exists c_z : (x_i, x_j) \in c_z$$

They represent the clustering and the real world scenario. If the two are true, we find a true positive, if they are both false a true negative. In the case the first one is true and the second is not, we have a false positive, a false negative otherwise.

We can observe that choosing a small number of clusters leads to a large quantity of false positives, while a fine grain partition, i.e. a high number of clusters, produces a high number of false negatives.

**Measures** We can use some measures of quality in clustering, for instance

$$Rand = \frac{TP + TN}{TP + TN + FP + FN}$$

but also *Purity* and *Normalized Mutual Information*.

## 5.2 K-means

This approach starts with  $k$  random centroids and assign each data point to the closest one. We then recompute the centroids with the current clusters and repeat the first point until termination.

The termination criterion could be a fixed number of iterations, no change in documents assignments or  $RSS$  measure below a threshold:

$$RSS = \sum_{k=1}^K \sum_{\vec{x} \in \omega_k} |\vec{x} - \vec{\mu}(\omega_k)|^2$$

where the measure is the sum of the distances between a point and its centroid squared. Note that at each iteration the  $RSS$  decrease.

Two issues of this approach is that the initial position of clusters changes the results a lot, and also, the  $k$  parameter is crucial to the algorithm. A good value for  $k$  is  $K = \min_K (\min(RSS_K) + \lambda K)$

### 5.3 Model-based clustering

A k-means generalization is obtained by interpreting the centroids as a model that generates data. A centroid with some added noise can generate a document.

**Idea** Instead of generating classes, we start from the points, we assume that they were generated with a generative model and we estimate the latent model parameters.

**Latent parameters**  $\Theta = \{\vec{\mu}_1, \dots, \vec{\mu}_n\}$  are the centroids to be found by k-means.

$L(D|\Theta)$  is the log-likelihood that the data  $D$  was generated by  $\Theta$ , this is the objective function, so  $\Theta$  becomes:

$$\Theta = \max_{\Theta} L(D|\Theta) = \max_{\Theta} \sum_{n=1}^N \log P(d_n|\Theta)$$

That means maximizing the sum of probabilities that documents are generated by the model. We are in a soft clustering scenario, as we are basically dealing with probabilities as cluster assignments.

#### 5.3.1 Expectation Maximization algorithm

This is an iterative algorithm that maximizes  $L(D|\Theta)$ , but can also be used to find latent models in a variety of applications.

Let's consider a multivariate Bernoulli distribution for data, so all documents are binary vectors, we want to estimate the probability that a given document is assigned to a specific cluster given the models parameters.

$$P(d|w_k; \Theta) = \prod_{t_m \in d} q_{mk} \cdot \prod_{t_m \notin d} (1 - q_{mk}) \quad \Theta_k = (\alpha_k, q_{1k}, \dots, q_{Mk})$$

where  $q_{mk} = P(U_m = 1 | \omega_k)$  is the probability that a document from the cluster  $\omega_k$  contains the term  $t_m$ .

$\alpha_k$  is the prior of the cluster  $k$ , so the probability that of a document to be in that cluster, not having any information about it.

**Process** We start by generating a document by picking a cluster  $w_k$  with probability  $\alpha_k$ , generating terms with probability  $q_{mk}$ .

We need to estimate the two parameters, and we do that iteratively similarly to k-means. The iteration is composed by two steps, *Maximization* and *Expectation*.

**Maximization step** The goal here is to estimate the parameters of the model, so the prior and the terms probability given the cluster:

$$q_{mk} = \frac{\sum_{n=1}^N r_{nk} I(t_m \in d_n)}{\sum_{n=1}^N r_{nk}} \quad \alpha_k = \frac{\sum_{n=1}^N r_{nk}}{N}$$

where  $I$  can be one or zero whether the term appears or not in the document, and  $r_{nk}$  is the soft assignment of the document  $n$  to the cluster  $k$ .

**Expectation step** We now estimate the probability of each term to be assigned to a cluster.

$$r_{nk} = \frac{\alpha_k (\prod_{t_m \in d_n} q_{mk}) (\prod_{t_m \notin d_n} (1 - q_{mk}))}{\sum_{k=1}^K \alpha_k (\prod_{t_m \in d_n} q_{mk}) (\prod_{t_m \notin d_n} (1 - q_{mk}))}$$

this is basically, for each cluster, the prior, multiplied by the product of  $q_{mk}$  or  $1 - q_{mk}$  for each term in the document computed before, normalized by the same quantity over all the clusters.

**Why do EM works?** The main idea of the algorithm is to push documents that contains the same words in the same cluster. Basically if two words appears in a document they should have a high probability of being assigned to the same cluster, so they have the same  $q_{mk}$ . This leads to assigning documents with the same words in the same clusters.

## 5.4 Affinity propagation clustering

**Input** The input is a similarity matrix, for instance  $s(i, j) = -||\vec{i} - \vec{j}||^2$ . There are also some special values for the diagonal of the matrix, where larger values are more likely to be chosen as exemplars for clusters.

**Idea** Documents exchange messages to decide exemplars and clusters assignments. This approach is similar to *page rank with authorities*<sup>1</sup>.

<sup>1</sup><https://nlp.stanford.edu/IR-book/html/htmledition/hubs-and-authorities-1.html>

**Responsibility messages** A message  $r(i, k)$  that denotes how well  $k$  is an exemplar for  $i$ . An example can be

$$r(i, k) = s(i, k) - \max_{k' \text{ s.t. } k' \neq k} \{a(i, k') + s(i, k')\}$$

basically we take the similarity of a pair of nodes and subtract the maximum similarity for any other document. This value will be zero for the most similar document, and negative for all other, as the quantity  $a(i, k')$  is initially zero.

**Availability** This message  $a(i, k)$  represent how appropriate is for  $i$  to choose  $k$  as exemplar.

$$a(i, k) = \min \left\{ 0, r(k, k) + \sum_{i' \text{ s.t. } i' \notin \{i, k\}} \max\{0, r(i', k)\} \right\}$$

**Self availability** The availability measure given the same document as argument, is

$$a(k, k) = \sum_{i' \text{ s.t. } i' \neq k} \max\{0, r(i', k)\}$$

**Process** At first, documents are connected by similarity, then we recompute the availability with respect to the responsibility and vice versa. This leads to making documents connections stronger and stronger and to the election of an exemplar for clusters. After some iterations convergence is reached.

## 5.5 Hierarchical clustering

This approach produces a hierarchy of clusters. It does not need the number of clusters but a criterion of optimal cluster selection. Most of this algorithms are deterministic.

**Process** Staring from a similarity matrix for the documents, we start taking the pair of documents with maximum similarity and group them in the same cluster. An idea is to have documents as nodes, and to add a cluster node connected to the two most similar documents.

The next step is to delete the rows and columns of the two selected documents, as they are already assigned, but, we add a new row and column for the cluster node. The similarity value between a document and a cluster is the minimum similarity in that cluster for the considered document, but many strategies can be applied.

After rebuilding the table, we continue with the initial step, note that if a cluster and document are selected as most similar items, we are adding a cluster connected to them, ergo we are considering the cluster obtained in the first step, as a sub-cluster for the last one.

The last step is to select the number of clusters. The previous logic created tree, as we grouped documents and clusters by adding nodes connected to them. We can choose a threshold for the intra-cluster similarity, or select a certain number of clusters, or considering the maximum jump in similarity in the obtained cluster tree and cut off the tree there. By cutting off the tree, we take the lower part, so the one with the leaves.

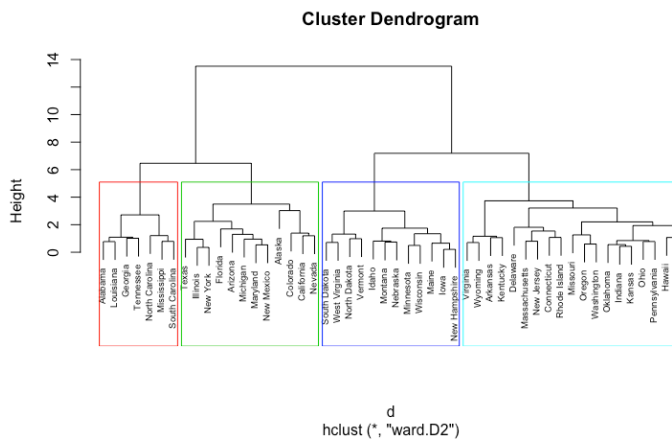


Figure 2: Hierarchical clustering example with 4 clusters

**Top-down clustering** This approach starts from a cluster containing all the documents, and recursively splits it keeping track of the parent of each document. The process stops when a threshold of intra-cluster similarity is obtained or when there's a cluster for each document. This approach is the opposite of the bottom up tree creation.

## 6 Topic modeling

**Vector space models problems** The approaches seen so far did not take into consideration semantics of words. We understood that the vector space models suffer from high dimensionality, the terms also could be synonyms, leading to ambiguity, in fact two words that means the same thing are two different dimensions in the model.

## 6.1 Linear algebra recap

**Eigenvalues and eigenvectors** Given a square matrix of size  $n$ , if we find a  $n$  dimensional vector such that

$$A\vec{x} = \lambda\vec{x}$$

$\lambda$  is a eigenvalue and  $\vec{x}$  is an eigenvector. It also holds that

$$(A - \lambda I)\vec{x} = 0 \implies |A - \lambda I| = 0$$

where  $I$  is the identity matrix, and the implication holds as  $\vec{x}$  is non-zero. In general the second part of the implication has  $n$  solutions, so  $n$  eigenvalues associated with a certain eigenvector.

After finding the solutions we can write that

$$\begin{aligned} AX &= A[\vec{x}_1, \dots, \vec{x}_n] = A\vec{x}_1 + \dots + A\vec{x}_n \\ &= \lambda_1\vec{x}_1 + \dots + \lambda_n\vec{x}_n = \Lambda X \end{aligned}$$

Where  $\Lambda$  is the diagonal matrix of the eigenvalues, and  $X$  the matrix with eigenvectors.

**Diagonalization** If the  $n$  eigenvector found at the previous step are independent, and the matrix  $A$  is invertible, which is not a problem as its square, we can write

$$AX = X\Lambda \rightarrow AXX^{-1} = X\Lambda X^{-1} \rightarrow A = X\Lambda X^{-1}$$

This representation is called diagonalization of  $A$ . Also, if  $A$  is symmetric, we can write

$$A = X\Lambda X^\top$$

## 6.2 Latent Semantic Indexing

The goal is to discover topics that motivate data by using matrix factorization techniques, while taking in consideration the possibility of expressing the same topic with different words.

**Idea** We learned that if a matrix is symmetric we can write the diagonalization as  $A = X\Lambda X^\top$ . The question is, what happens if we multiply the document-term matrix by itself?

We find a symmetric matrix that somehow encapsulates a notion of similarity of documents because if two documents share the same words the product will be really high for that two documents.

If we multiply the term-document matrix we obtain a similar result.

**Process** Formally, we consider a matrix  $B_{n \times n}$  as the product of the document-term matrix  $A_{m \times n}$  by itself. We then take the eigenvalues of the matrix  $B$  and start diagonalization.

The process involves first sorting the eigenvalues, and taking the first  $k$  non-zero ones, that will be  $k$  topics we are finding in the documents, and we arrange the corresponding eigenvectors in  $V = [\vec{v}_1, \dots, \vec{v}_k]$ .

We proceed with the definition of  $U = [\vec{u}_1 = \frac{1}{\sigma_1} A \vec{v}_1, \dots, \vec{u}_k = \frac{k}{\sigma_k} A \vec{v}_k]$ , that contains perpendicular  $m$ -dimensional vectors of length 1. Then we can write:

$$A = U \Sigma V^\top$$

where  $\Sigma_{k \times k}$  is the diagonal matrix having  $\sigma_i = \sqrt{\lambda_i}$  along the diagonal.

**Singular value decomposition** A more simple way of seeing the vectors introduced in the previous paragraph is:

- $V$  are the eigenvectors of  $AA^\top$
- $U$  are the eigenvectors of  $A^\top A$
- $\Sigma$  is the diagonal matrix composed starting from the eigenvalues of  $AA^\top$  and  $A^\top A$ , which are the same

**Defining topics** After applying singular value decomposition to the document term matrix we have the latent topic matrix  $\Sigma_{k \times k}$ . This matrix somehow encapsulate the top  $k$  relevant topics for the documents, and now, to make them human understandable we assign each a set of words, and also, we can assign to each document a topic, in a soft clustering fashion.

Basically, given the three matrixes that SVD outputs, namely  $U, \Sigma, V$ , we can discover what topics have the documents by multiplying  $U$  and  $\Sigma$ . The output will be a matrix with document on rows and topics on columns, where the absolute value of a cell represent how strong a topics is present in a document.

In the same way we can find a relation between topics and words by multiplying  $\Sigma$  and  $V^\top$ .

### 6.3 Latent Dirichlet Allocation

This approach relies on probability theory to find topics in documents. In the LSI approach we tried to find a *glue*, formally  $\Sigma$ , between document and terms.

In LDA the goal is to estimate the probabilistic relation between topics and documents, basically the probability that a given document contains a given topic, and the one that a topics contains a certain word.