Text Mining and Search

Alberto Filosa

29/9/2020

Indice

1 Introduction **2**
1.1 Basic Text Processing 2

2 Text Processing **2**

3 Text Representation **3**
3.1 Natural Language Processing 4
3.1.1 Part Of Speech Tagging 4
3.1.2 Named Entity Recognition 4
3.2 Language Models 5

4 World Embedding **6**
4.1 Count-Based Models 7
4.2 Predictive Models 8

5 Text Classification **8**
5.1 Supervised Learning 9
5.2 Supervised Classification 9
5.3 Evaluations 9

6 Text Clustering **10**
6.1 Flat Clustering 10
6.2 Hierarchical Clustering 10
6.3 Clustering Evaluation 11

7 Topic Modeling **11**
7.1 Latent Semantic Analysis 11
7.2 Latent Dirichlet Analysis 11
7.3 Evaluation 12

8 Topic Classification **12**

9 Text Summarization **12**
9.1 Evaluation 13

10 Information Retrieval **13**
10.1 Indexing 14
10.2 Matching 14
10.2.1 Boolean matching 14
10.2.2 Vector Space matching 14

11 Web Search **14**
11.1 Updating documents: Gathering 15
11.2 Ranking 15
11.2.1 Pagerank 15
11.2.2 HITS 15

12 Recommender Systems **15**
12.1 Collaborative filtering 16
12.2 Content-based filtering 16
12.3 Knowledge-based filtering 16

1 Introduction

> Text mining is a burgeoning new field that attempts to glean meaningful information from natural language text. It may be loosely characterized as the process of analyzing text to extract information that is useful for particular purposes.

Text mining is generally used to denote any system that analyzes large quantities of text and detects lexical or linguistic patterns extracting useful information.

It is generally used for sentiment analysis, document summarization, news and other recommendation and text analytics.

Data mining can be more characterized as the extraction of implicit, unknown, and potentially useful information from data. With text mining, the information extracted is clearly and explicit in the text.

There are several problems about Text Mining: first, document in an unstructured form are not accessible to be used by computers, many data are not well organized and natural language text contains ambiguities on a lexical, syntactical and semantic levels.

The Information Retrieval make it easier to find things on the Web because it finds useful answers in a collection.

The main tasks affected by Text Mining are:

- *Text Summarization*, that produce a condensed representation of the input text;
- *Information Retrieval*, that identifies the most relevant documents to the query (used in Web Search Engine);
- *Content Based Recommender System*, that define the user profile and suggests the affinity of the research;
- *Text Classification*, that define categories according to their content (e.i. Sentiment Analysis);
- *Document Clustering*, an unsupervised learning in which there is no predefined classes, but only groups of document that share the similar topics.

1.1 Basic Text Processing

Basic Text Processing is often the first step in any text mining application. It consists of several layers of simple processing such as tokenization, lemmatization or normalization. The purpose of analyzing texts can be either predictive or exploratory.

Predictive Analysis develope computer programs that automatically detect a particular concept within a span of text. Main examples are:

- Opinion Mining automatically detect if a sentence is positive or negative;
- Sentiment Analysis automatically detect the emotional state of the author in a text;
- Bias detection automatically detect if an author favors a particular viewpoint;
- Information Extraction automatically detect that a short sequence of words belongs to a particularly entity type;
- Relation Learning automatically detect pairs of entities that share a particular relation;
- Text Driven Forecasting monitors incoming text and making prediction about external events or trends;
- Temporal Summarization monitors incoming text about a news event and predicting whether a sentence should be included in an on-going summary of the event.

Exploratory Analysis develop computer programs that automatically discover interesting and useful patterns in text collections.

A document is a list of text, structure, other media such as images and sounds, and metadata. *Metadata* associated with a document is data related to the document and it consists of descriptive, related to the creation of the document, and semantic metadata, relate to the subject dealt with in the document.

Metadata is information about information, cataloging and descriptive information structured to allow pages to be properly searched by computers.

2 Text Processing

The aim of *Text Processing* is extract and identifying the corpus of the text. When defining a collection, it is necessary read the document, so make it processable and understand the worlds in a semantic level. Terms are the basic features of text and they collected in a bag-of-world.

Tokenization is the process of demarcating and possibly classifying sections of a string of input characters. A token is an instance of a sequence of characters, so each of these is a candidate to be a meaningful term after further processing. There are many problems about this procedure: firstly, how split a city name like `San Francisco`

or `Los Angeles`; secondly, there are some problem tokenizing hyphenated sequence; numbers, recognized in different terms, and language issues (like German or Arabic language). Basically, the tokenizations consists in breaking a stream of text into meaningful units that depends on language, corpus or even context.

```
It's not straight-forward to perform so-called
"tokenization"
```

`It`, `'`, `s`, `not`, `straight`, `-`, `forward to`, `perform`, `so`, `-`, `called`, `"`, `tokenization`, `.`

To solve these problem is useful use Regular Expression or Statistical Method to detect the token in a specific sentence. They explore rich feature to decide where the boundary of a word is.

A ***Stop Word*** is a word, usually one of a series in a stop list, that is to be ignored by a Search Engine (e.g. `the`, `a`, `and`, `to`, `be`, etc.). However, Web SE do not use stop lists due to the good compression techniques and needed in most of search queries, such as song titles of relational queries.

The ***Normalization*** is the process of mapping variants of a term to a single, standardized form. It may depends on the language an so is intertwined with language detection. In Search Engines is necessary to normalizes indexed text as well as query terms identically.

Case folding is the process reducing all letters to lower case. There are some exception, such as proper noun or acronymous, but often the best choice is to lower case everything.

Thesauri is the process to handle synonyms and homonyms in text sentences. It can be handled by hand-construct equivalence classes, rewrite the word to form equivalence-class terms or in Search Engines it can expand a query.

Soundex is an approach which forms equivalence classes of words based on phonetic heuristic, useful for spelling mistakes.

Lemmization is the process to reduce variant forms to base form:

`am`, `are`, `is` → `be`

`car`, `cars`, `car's`, `cars'` → `car`

It implies doing proper reduction to dictionary headword form. A crude affix chopping is ***Stemming***, that reduce inflected or derived words to their root form:

`automate`, `automatic`, `automation` → `automat`

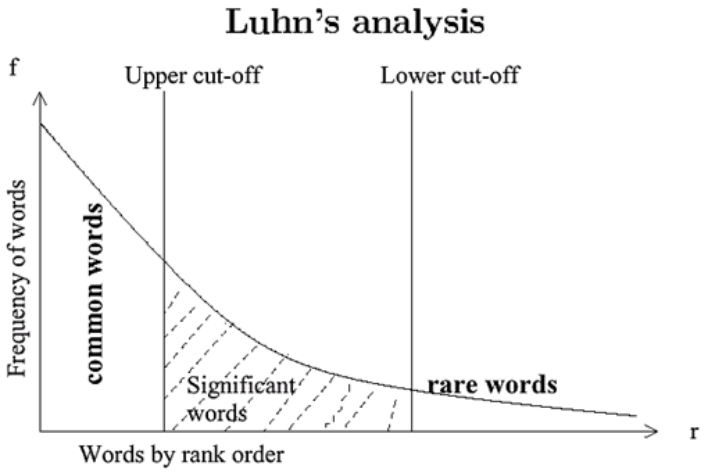
3 Text Representation

Text Representation is the process to representing the text with graphical method. The simplest way is the binary term-document weighting, in which each document can be represented by a set of term (or binary vector) $\in \{0, 1\}$. Another graphical method is the count matrix, that consider the number of occurrences of a term in a document, but this not consider the ordering of words in a document.

In natural language, there are few frequent terms and terms; the *Zipf's Law* describes the frequency, $f(w)$, of an event in a set according to its rank, approximately constant (usually in an increasing order).

$$f(w) \propto \frac{1}{r(w)} = \frac{K}{r(w)}$$

In particular, head words take large position of occurrences, but they are semantically meaningless (e.g. `the`, `a`, `we`), while tail words take major portion of vocabulary, but they rarely occur in documents (e.g. `dextrosinistral`).



According to this *Luhn's Analysis*, words do not describe document content with the same accuracy: word importance depends not only on frequency but also on position. So frequencies have to be weighted according to their position: two cut-offs are experimentally signed to discriminate very common or very rare words. Most significant words are those between the two cut-offs, those that are neither common nor rare. It automatically generates a document specific stop list, the most frequent words.

Weights are assigned according to corpus-wise, carrying information about the document content, and document-wise, so not all terms are equally important.

The *Term Frequency* $tf_{t,d}\}$ represent the frequency of the term t in the document d, often normalized by document length:

$$w_{t,d} = \frac{tf_{t,d}}{|d|}$$

To prevent bias towards longer documents, it doesn't consider the document length but the frequency of the most occurring word in the document:

$$w_{t,d} = \frac{tf_{t,d}}{\max\limits_{t_i \in d}\{tf_{t_i,d}\}}$$

The *Inverse Document Frequency* idf_t represents the inverse of the informativeness of the document for a term t:

$$idf_t = \log\left(\frac{N}{df_t}\right)$$

df_t is the number of documents that contains t and N is the total number of documents.

The *Weights*, called tf-idf weights, are the product of the two indices:

$$w_{t,d} = \frac{tf_{t,d}}{\max\limits_{t_i \in d}\{tf_{t_i,d}\}} \cdot \log\left(\frac{N}{df_t}\right)$$

The weight w increases with the number of occurrences within a document and with the rarity of the term in the collection.

3.1 Natural Language Processing

Text representation can be enriched using ***N***atural ***L***anguage ***P***rocessing (**NPL**), which provides models that go deeper to uncover the meaning. A real problem about Text processing is how phrases are recognized. There are 3 possible solutions:

- Use word N-grams;
- Identify the syntactic role of words within phrases using a Part-Of-Speech tagger;
- Store word positions in indexes and use proximity operators in queries.

3.1.1 Part Of Speech Tagging

The ***P***art ***O***f ***S***peech (**POS**) Tagging is a procedure that assigns to each word its syntax role in a sentence, such as nouns, verbs and adverbs. There are many applications:

- Parsing;
- Word prediction in speech recognition;
- Machine Translation;

The POS Tagging problem is determine the POS tag for a particular instance of a word, because words often have more than one tag, like `back` (*the back door*, *on my back*). It is possible to assume a conditional probability of words $\mathbb{P}(w_n|w_{n-1})$ or POS likelihood $\mathbb{P}(t_n|t_{n-1})$ to disambiguate sentences and assessing the likelihood.

1. Start with a dictionary;
2. Assign all possible tags to word from the dictionary;
3. Write rules to selectively remove tags;
4. Leave the correct tag for each word.

3.1.2 Named Entity Recognition

The ***N***amed ***E***ntity ***R***ecognition (**NER**) is a very important task of Information Extraction because it identifies and classifies names in text in a particular application. It involves identification of proper names in text and classification into a set of predefined categories of interest.

Category definitions are intuitively clear, but there are many gey areas caused by metonomy, for example Organization (`England won the World Cup`) versus Location (`The World Cup took place in England`).

NER has different approaches:

- *Ruled-Based*, identifying if a sequence of words can be an entity name using lexicons that categorize names (developed by Trial and Error or ML techniques);
- *Statistic-Based*, maximizing the probability of an entity using Hidden Markov Model estimated with training data and resolving ambiguity using context.

HMM describes a process as a collection of states with transitions between them. Each state is associated with a probability distribution over worlds of possible outputs. For identifying named entities, it finds sequence of labels which gives the highest probability of the sentence.

Natural Language is designed to make human communication efficient, but people omit a lot of common sense knowledge and keep lots of ambiguities, such as Word-Level or Syntactic ambiguity.

Usually, a message is sent by the sender and received by the recipient, who infers *Morphological* information of the words, uses lexical rules (*Syntax*) to reconstruct the original message and compare it with its knowledge to understand its *Semantic*, what is explicit, and does not change with the context, and *Pragmatic*, what is implicit or context related. It is the study of the relationship between language and context, fundamental to explain the understanding of the language. In the same way, a program have to rebuild the message and compare it whit a base of knowledge (usually modeled as a graph) to understand its content (*Inference*) handling a large chunk of text (*Discourse*).

The Context is the situation in which the communicative act takes place and the set of knowledge, beliefs and the like are shared by both people. It includes at a minimum the beliefs and assumptions of the language users, relating to actions, situations and a state of knowledge and attention of those who partecipate to social interactions.

To build a computer that understands text it is necessary thus NLP Pipeline:

1. Syntactic Parsing, a grammatical analysis of a given sentence, conforming to the rules of a formal grammar;
2. Relation Extraction, identifying the relationship among named entities;
3. Logic Inference, which converts chunks of text into more formal representation.

3.2 Language Models

A *Language Model* (**LM**) is a formal representation of a specific language, generating a probability distribution of the words in topics. A probability to a sequence words can be used for:

- Spell Correction;
- Speech Recognition;
- Text Categorization;
- Information Retrieval;
- Summarization.

Main tasks are computing the probability of a sentence (or a sequence of words), $\mathbb{P}(W) = \mathbb{P}(w_1, w_2, \dots, w_n)$ and the probability of an upcoming word: $\mathbb{P}(w_{n+1}|w_1, \dots, w_n)$. Any model which computes any of these procedure is called a Language Model.

The joint probability is computed relying on the *Chain Rule* of probability:

$$\begin{aligned}\mathbb{P}(w_1, w_2, \dots, w_n) &= \mathbb{P}(w_1)\mathbb{P}(w_2|w_1) \dots \mathbb{P}(w_n|w_1, \dots, w_{n-1}) \\ &= \prod_{i=1}^{n} \mathbb{P}(w_i|w_1, \dots, w_{i-1})\end{aligned}$$

but so many product could lead the number to decay pretty fast. A solution is the *Markov Assumption*, an approximation of the joint probability in which considers the probability of one or two words preceding the word:

$$\mathbb{P}(w_1, w_2, \dots, w_n) \approx \prod_{i=1}^{n} \mathbb{P}(w_i|w_{i-k} \dots w_{i-1})$$

In this case is possible to construct an *Unigram Language Model*, the probability distribution over the words in a language, or *N-gram Language Model*, where probabilities depend on previous words. For example, a Bigram LM is the probability of a word in a sequence where considering the previous word. The Maximum Likelihood Estimate of a Bigram Language Model is the following:

$$\mathbb{P}(w_i|w_{i-1}) = \frac{c(w_{i-1}, w_i)}{w_{i-1}}$$

For example, look at this repository:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I do not like green eggs and ham </s>
```

The probability of `I` at the start of the sentence is $\mathbb{P}(I| < s >) = 2/3 = 0.67$, while the probability of Sam in as the last word is $\mathbb{P}(Sam| < s >) = 1/3 = 0.33$.

The evaluation of a Language Model is the same as most important models: the dataset is splitted in Training set and test set, Testing the model's performance on unseen data. The best evaluation for comparing models is compare the accuracy of those. An important metric evaluation is *Perplexity*, the inverse probability of the test set normalized by number of word (minimizing perplexity is the same as maximizing probability):

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{\mathbb{P}(w_i|w_1 \dots w_{i-1})}}$$

4 World Embedding

World Embedding indicates a set of techniques in a NLP where words from a vocabulary are mapped to vectors of real numbers. It involves a mathematical embedding from a space with many dimensions per word to a vector space with a lower dimension. This model can be used to check similarity between either or words and check topic similarity and words usage per topic (worlds can be represented as vectors too).

Each document is represented by a vector of words, with a binary representation, the frequency of the words or the weighted representation (TF-Idf).

Usually, the Similarity is not computed by using term-document representation, because two words are similar if their context vectors are similar. To represent words as vectors, it is possible to build a $V \times V$ *1-hot* matrix (also known as *Local Representation*) in which each word has all values equal to 0 except for the corresponding colum, equal to 1. This is a very sparse matrix, with no information between similar words. To solve these problems, the idea is associate k context dimensions representing properties of the words in the vocabulary, also known as *Distributed Representation*. Distributed vectors allow to group similar words together depending on the considered context.

The difference between Local and Distributed representation is in the matrix: in particular, in LR every term in a vocabulary T is represented by a binary vector, while in DR every term is represented by a real-valued vector.

For simple scenarios it is possible create a k-dimensional mapping for a simple example vocabulary and manually choosing contextual dimensions that make sense. This is not a simple task, in particular manual assignment of vector would be impossibly. The *Distributional Hypothesis* aims to represent each words by some context. It is possible to use different granularities of context, such as documents and sentences.

The *Window-based Co-occurence Matrix* counts the number of times each context words co-occurs inside a windows of a particular size with the word of interest. It generates an $X = |V| \times |V|$ co-occurence matrix. The frequency is not a good representation, it is possible to use weights, TF-IDF or ***P***ointwise ***M***utual ***I***nformation (**PMI**):

$$PMI(w_1, w_2) = \log_2 \frac{\mathbb{P}(w_1, w_2)}{\mathbb{P}(w_1)\mathbb{P}(w_2)} \in (-\infty, +\infty)$$

To have better estimation, only positive estimation are taken:

$$PPMI(x, y) = \max[PMI(x, y), 0]$$

$f_{i,j}$ is the number of time the word w_i appears in the context c_j. It is used to compute these probabilities:

$$\mathbb{P}(w_j, c_j) = \frac{f_{i,j}}{\sum_{i,j=1}^{W,C} f_{i,j}}$$

$$\mathbb{P}(w_j) = \frac{\sum_{i=1}^{C} f_{i,j}}{\sum_{i,j=1}^{W,C} f_{i,j}}$$

$$\mathbb{P}(c_j) = \frac{\sum_{i=1}^{W} f_{i,j}}{\sum_{i,j=1}^{W,C} f_{i,j}}$$

For example, look at this table:

```
##             Computer Data Pinch Result Sugar
## Apricot            0    0     1      0     1
## Pineapple          0    0     1      0     1
## Digital            2    1     0      1     0
## Information        1    6     0      4     0
```

The $\mathbb{w}_4, \mathbb{c}_2 = 6/19 = 0.32$, $\mathbb{w}_4 = 11/19 = 0.58$ and $\mathbb{c}_2 = 7/19 = 0.37$. The PPMI table is:

```
##             Computer Data Pinch Result Sugar
## Apricot         0.00 0.00  0.05  0.000  0.05
## Pineapple       0.00 0.00  0.05  0.000  0.05
## Digital         0.11 0.05  0.05  0.050  0.00
## Information     0.05 0.32  0.00  0.021  0.05
```

Rare words produce higher scores, that risks to bias the analysis: probabilities of w and c are modified or a k-smoothing is used to reduced those scores with a value $\alpha \in [0, 1]$ (usually $\alpha = 0.75$):

$$PMI_\alpha(w, c) = \log_2 \frac{\mathbb{P}(w, c)}{\mathbb{P}(w)\mathbb{P}_\alpha(c)} \in (-\infty, +\infty)$$

$$\implies PPMI_\alpha(w, c) = \max[PMI_\alpha(w, c), 0]$$

with

$$P_\alpha(c) = \frac{\text{count}(c)^\alpha}{\sum_c \text{count}^\alpha}$$

A Co-occurrence Matrix in reality is constituted by a very large number of words. For each word, Tf-Idf and PPMI vectors are long ($|V| \approx 50'000$) and sparse (most elements = 0). There are techniques to learn lower-dimensional vectors for words, which are Short and Sense. These dense vectors in a latent space are called *Embedding*.

4.1 Count-Based Models

Count-Based Models computes the statistic of how often some words co-occurs with its neighbor words in a large text corpus. Then it is possible to reduce the dimensionality the vector in an small and dense array for each word. The lower-dimensional matrix of words and features represent a dense vector for each word.

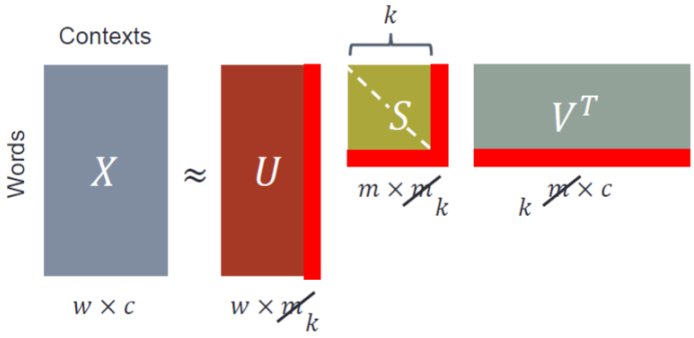
The most popular model is the ***S**ingular **V**alue **D**ecomposition* (**SVD**), a dimensionality reduction techniques used in Text Mining. Any rectangular $w \times c$ matrix X can be expressed as the product of 3 matrices:

$$\underset{w\times c}{X} = \underset{w\times m}{U} \times \underset{m\times m}{S} \times \underset{m\times c}{V^t}$$

where:

- U is an $w \times m$ matrix with w words of the original X matrix with m features in a new latent space;
- S is an $m \times m$ diagonal matrix of *Singular Values* expressing the importance of each dimension;
- V^t is an $m \times c$ matrix with c features of the original matrix X and m singular values rows.S

It is possible to select the top-k singular values obtaining a low rank approximation of the original matrix X. Instead of multiplying these matrices, it is useful to make only use of the matrix U. Each row of U contains a k-dimensional vector representing a word in the vocabulary.



$$\underset{w\times c}{X} \approx \underset{w\times k}{U} \times \underset{m\times k}{S} \times \underset{k\times c}{V^t}$$

Unfortunately, there are many drawbacks regarding this method. First of all, the *Dimension* of the matrix *Change* very often: new words are added very frequently and corpus changes in size. Secondly, the matrix is extremely *Sparse* since most word do not occur and it is necessary to construct a quadratic loss function to perform an SVD. To resolve these problems there are some hacks on X obtaining more satisfying values: ignoring *Stop Words*, applying a *Ramp Window*, a co-occurrence count based on the distance between the words in the document or use the *Pearson Correlation* and set negative counts to 0 instead of using PPMI.

Another important Count-Based model is the ***Glo**bal **Ve**ctors for Word Representations* (**GloVe**), a model that leverages statistical information by training only non-zero element in a word-word co-occurrence matrix, rather than on the entire sparse matrix (SVD) and on individual context windows in a large corpus (Word2Vec). In particular, global corpus statistics are captured directly by the model.

In the glove model, instead of considering just the p of the context word appearing, we are going considering the ration of the probability of the words. Only in the relation capture non discriminative words because large values (> 1) correlate well with properties specific to a word and

The ration P_{ik}/P_{Jk} depends on three words i, j and k. The general form of the glove model is useful to learn the word vectors representation:

$$F(w_i, w_j, \tilde{w_k}) = \frac{P_i k}{P_j k}$$
$$w_i^t \tilde{w_k} + b_i + \tilde{b_k} = \log(X_{ik})$$

where $w \in R^d$ are word vectors and $\tilde{w_k} \in R^d$ are separate context word vectors.

The GloVe model builds an objective function J associating word vectors to text statistics using the Least Square Error with a weighted function $f(X_{ik})$:

$$J = \sum_{i,k=1}^{V} f(X_{ik})(w_i^t w_k + b_i + b_k - \log(X_{ik})$$
$$= \frac{1}{2} \sum_{i,k=1}^{V} f(X_{ik})(w_i^t \tilde{w_k} - \log(X_{ik}))^2$$

where V is the size of the vocabulary.

The two sets of vectors captures similar co-occurence information. The best solution is to simply sum them up:

$$X_{final} = U + V$$

4.2 Predictive Models

Predictive Models predict a word from its neighbors obtaining a small, dense embedding vectors. The most popular models is the *Word2Vec* model, which creates collections of similar concept automatically on raw texts (supervised training data) and without advanced language skills. Very good performance are obtained by employing very large ($\approx$ 10M words) text including as many different words as possible.

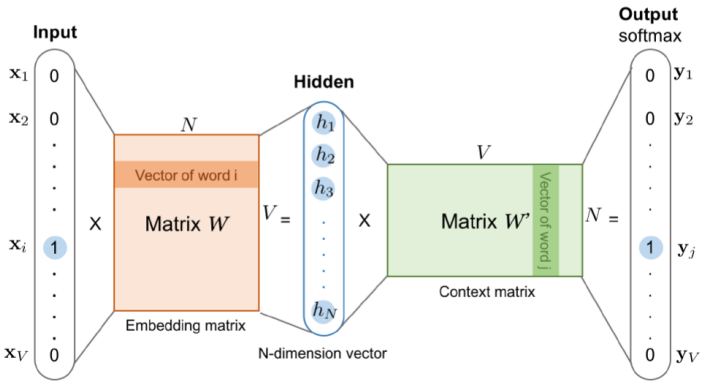
It is similar to Language Modelling, but it predicts the context rather than next word. The loss function is the following:

$$J = 1 - P(\text{context} \mid \text{word})$$

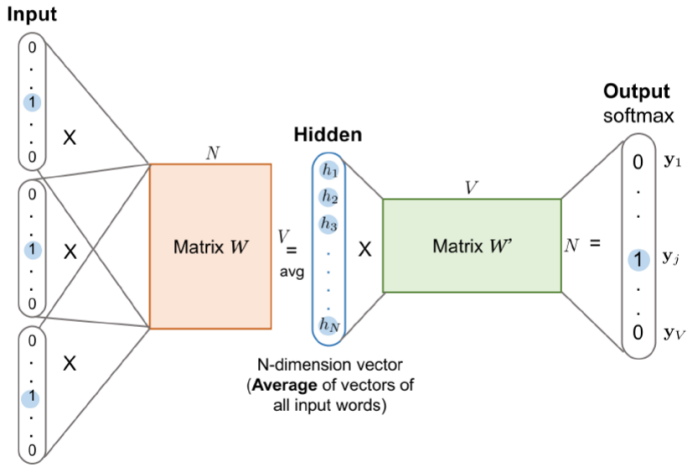
There are two training method:

- Hierarchical Softmax;
- Negative Sampling.

The *Skip Gram* model predicts the surrounding (context) words based on the current (centered) word. Given a sliding window of a fixed size moving along a sentence, the word in the middle is the target, while near words (in the sliding window) are context words. Then it is trained to predict the probabilities of a word being a context word for the given target. The model is a Neural Network with just one Hidden Layer representing the word embedding of size N. The Input layer x and the Output Layer y are one-hot encoded word representations.



The ***C***ontinuos **B**ag ***O***f ***W***ords (**CBOW**) is another similar model learning for learning word vectors. It predicts the target word from source context words.



While the Skip Gram model works well with a smaller training dataset because it represents well even rare words, the CBOW is several times faster to training the model and has a slight better Accuracy for frequent words.

Predictive models can capture complex pattern beyond word similarity and generates improved performance on other tasks, but it scales with corpus size (it is necessary to train every time the data) and it does not use efficient statistics.

5 Text Classification

Text Classification is the activity of predicting which data items belongs to a predefined finite set of classes. In Text Classification, data items can be textual (E.g: news articles, e-mail, etc.) or partly textual (web pages).

$$h : D \to C$$

There are many types of classification:

- *Binary Classification*, where each items belongs to exactly one classes in a set of two;
- *Single Label Multi Class Classification*, where each item belongs to exactly one classes in a set of many classes (> 2);
- *Multi Label Multi Class Classification*, where each item may belong to zero, one or several classes together;
- *Ordinal Classification*, similar to *SLMC*, but classification are ordered respectably to some quality.

The previous definitions denote hard classification (training a soft classifier with a score and picking a threshold

t). Soft classification aims to predict a score for each pair (d, c) where the score denotes the probability that d belong to c, where scores are used for ranking.

There are many applications:

- *Knowledge Organization*, conferring structure to an otherwise unstructured body of knowledge (E.g: classifying news articles);
- *Filtering*, blocking a set of non relevant items from a dynamic stream. It is a binary classification (E.g: spam filtering detecting unsuitable content);
- *Empowering*, improving the effectiveness of other tasks in Information Retrieval or Natural Language Process.

5.1 Supervised Learning

The Supervised Learning is a generic learning algorithm used to train a classifier from a set of manually classified examples. The classifiers learns the characteristics that a new text should have in order to be assigned to class c.

In order to generate a vector based representation for a set of documents, there are 3 main steps:

1. *Feature Extraction* by topic, a set of features that coincide with the set of words occurring in the training set (Unigram model). An alternative is make the set of features coincide with the set of character n-grams occurring in the document. The choice of features is different for classification tasks;
2. *Feature Selection*, identifying the most discriminative features discarding the others. The filter approach consists in measuring the discriminative power of each feature and retaining only the top-scoring features. A typical choice is Mutual Information, measuring the mutual dependence between two variables. It can be used also with matrix decomposition like SVD, PCA, aggregating different features:

$$MI(t_k, c_i) = \sum \sum \mathbb{P}(t, c) \log_2 \frac{\mathbb{P}(t, c)}{\mathbb{P}(t)\mathbb{P}(c)}$$

3. *Feature weighting* means attributing a value to feature in the vector that represents document. This value can be binary, numeric obtained in a unsupervised or supervised classes.

The classical way to represent documents is the vector space model, where documents are converted into vectors in a common vector space. It is constituted by terms (axes) and documents are points in the vector space. To represent the similarity of a document is possible to use the cosine similarity:

$$\mathbb{sim}(d_1, d_2) = \frac{\sum_{i=1}^{k} w_{i1} w_{i2}}{\sqrt{\sum_{i=1}^{k} w_{i1}^2 \sum_{i=1}^{k} w_{i2}}^2} \in [0, 1]$$

5.2 Supervised Classification

For binary classification, essentially any supervised learning algorithm can be used for training a classifier. One of the most important method is ***S***upport ***V***ector ***M***achine (**SVM**), which aims to find the separating surface that maximizes the margin between hyperplane and training points. Often classification problems are not linearly separable, but it can become linear separable once mapped to an high-dimensional space using kernels. The trained classifiers often depend on one or more parameters, but they need to be optimized with a k-fold cross-validation on the training set.

5.3 Evaluations

There two important aspects to evaluate a classifier: *Efficiency*, which refers to the consumption of computational resources (training and classification), while Effectiveness (*Accuracy*) refers to how frequently classification decisions taken by a classifier are correct. For binary classification it is possible to evaluate accuracy with confusion matrix:

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

Another measure is F_1, robust to the presence of imbalance in the test set. It is defined as the harmonic mean of Precision and Recall:

$$F_1 = \frac{\pi\rho}{\pi + \rho} = \frac{2TP}{2TP + FP + FN}$$

For multi-label multi-class classification, F_1 must be averaged across the classes according to micro-averaging, or macro-averaging.

For single-label multi-class classification, the most used measure is the Accuracy.

6 Text Clustering

The Document Clustering is the process of grouping a set of documents into classes of similar object. It is an unsupervised learning, so clusters are formed from data without human input. In particular:

- Object *within* a cluster should be similar;
- Object *from different* clusters should be dissimilar.

It is possible to use Cluster Analysis with documents because objects in the same cluster behave similarly with respect to relevance to information needs. There are different types of clustering structure:

- *Partitional*, a division of the set of data objects into non-overlapping subsets (clusters) such that each data object is in exactly one subset vs *Hierarchical*, sub-clusters organized as a tree.
- *Exclusive*, each data is assigned to a single cluster vs *Overlapping*, data can be assigned in more than one cluster.
- *Complete*, every data is assigned to a cluster vs *Partial*, where every data is not necessary assigned to a cluster.

6.1 Flat Clustering

Flat Clustering, known also as Prototype-Based Clustering, is a clustering algorithm in which any object is closer to the prototype that defines the cluster to which it belongs to than to the prototype of any other cluster.

The goal of this algorithm is compute an assignment $\gamma : D \to \{w_1, \dots, w_k\}$ that minimize the objective function (in term of similarity or distance) given a set of documents D, a desired number of clusters k. *k-means* defines a prototype in terms of a centroid, which is usually the mean (average squared Euclidean Distance) of the documents in the real-valued space. The k-means algorithm is defined as follows:

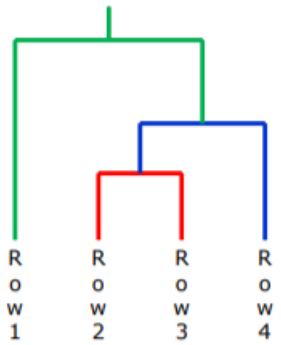
1. Select k objects as centroids;
2. Form k clusters and assign each document to closest centroid;
3. Compute the centroid for each cluster;
4. Go to **1.** until centroid do not change.

6.2 Hierarchical Clustering

Hierarchical Clustering is a clustering algorithm in which sub-clusters are organized as a tree. It also does not require to specify the number of clusters. It can be:

- *Agglomerative*, where all single objects are merged into the closest pair of cluster (by proximity);
- *Divisive*, where an all-inclusive cluster is splitted in different clusters until only single clusters of individual object remain using a flat clustering algorithm (it is necessary to specify which cluster to split and the cut of the dendrogram).

Hierarchical Clustering is often displayed graphically using a tree-like diagram called *Dendrogram*, which displays the sub-cluster relationships and the order in which the clusters were merged or splitted.



There are different ways to define the closest pair of cluster:

- *Single-link*, where the similarity of 2 clusters is the similarity of their most similar members (find the minimum distance between two clusters):

$$d(w_i, w_j) = min_{x \in w_i, y \in w_j} d(x, y)$$
$$d((w_i \cup w_j), w_k) = min[d(w_i, w_k), d(w_j, w_k)]$$

- *Complete-link*, where the similarity of 2 clusters is the similarity of their most dissimilar members (find the maximum distance between the elements):

$$S(w_i, w_j) = min_{x \in w_i, y \in w_j} S(x, y)$$
$$S((w_i \cup w_j), w_k) = min[S(w_i, w_k), S(w_j, w_k)]$$

- Group-average;
- Centroid.

6.3 Clustering Evaluation

It is difficult to evaluate the validation of a cluster algorithm. A good clustering will produce high quality clusters in which the external class similarity is high and the internal class similarity is low.

The most important *Internal Evaluation* index is the Silhouette Coefficient, that measures how well an observation is clustered and estimates the average distance between clusters. It is defined as follows:

$$S_i = \frac{b_i - a_i}{\max(a_i, b_i)} \in [-1, +1]$$

- a_i is the average distance of the i-th object to all other objects in its cluster;
- b_i is the minimum of the average distances of the i-th object to all the objects in each given cluster.

The most important *External Measure* is the Purity, defined as the ratio between the dominant class and the size of the cluster:

$$Purity(w_i) = \frac{1}{n_i} \max_j(n_{ij}) \in [0, 1]$$

Other external measures are:

- Rand Index, defined as the percentage of correct assigment to a cluster: $RI = \frac{TP+TN}{N} \in [0, 1]$;
- Precision, Recall and F-measure.

7 Topic Modeling

Topic Modeling is an unsupervised machine learning tecnique aimed at scanning a set of documents, detect words and phrases patterns within them and automatically clustering word groups which best characterize a set of documents. It provides a set of words that makes sense together (*topic*).

The main differences between Text Clustering and Topic Modelling is that the first procedure groups documents into different clusters based on similarity (or distance) measures, while the second one groups words into different clusters where each word have a probability of occurrence for the given topic.

7.1 Latent Semantic Analysis

The ***L***atent ***S***emantic ***A***nalysis (**LSA**) is a Topic Modeling Technique which computes how frequently words occur in the documents and assumes that similar documents will contain the same distribution of word frequencies. The main idea is decompose the Document Term Matrix into a Document Topic and Topic Term matrix.

The standard method for computing word frequencies is the Tf-Idf for each term in a document. It can be decomposed using SVD:

$$X = \underset{\text{Document Topic}}{U} \times \underset{\text{Potential Topic}}{S} \times \underset{\text{Term Topic}}{V^t}$$

The problem of dimensionality reduction is that cannot have an interpretable meaning in the natural language. Furthermore LSA can only capture partially polysemy, but this is not a problem because it can find the predominant sense in a document.

An alternative of this techniques is the *Probabilistic LSA* (**pLSA**), which identifies and distinguishes words in different contexts without using the dictionary. In particular, it allows to disambiguate polysemy and discloses topical similarities grouping together words sharing the same context.

It finds a probabilistic model with hidden (or latent) topic that can generate the observed data in the Document Term Matrix. pLSA expresses data in three different variables:

- *Documents* observed variables: $d = \{d_1, \dots, d_N\} \in D$;
- *Words* observed variables: $w = \{w_1, \dots, w_M\} \in W$;
- *Topics* hidden variables: $z = \{z_1, \dots, z_K\} \in Z$.

$$\begin{aligned} \mathbb{P}(D, W) &= \prod_{(d,w)} \mathbb{P}(d, w) \\ &= \sum_{Z \in z} \mathbb{P}(z)\mathbb{P}(d|z)\mathbb{P}(w|z) \\ \implies A &\approx U_k S_k V_k^t \end{aligned}$$

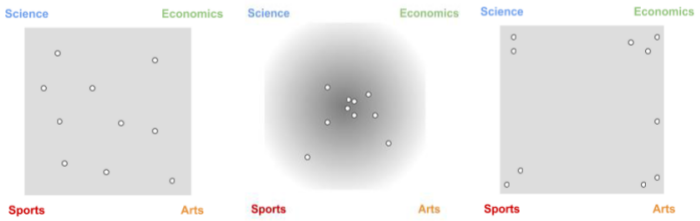
7.2 Latent Dirichlet Analysis

The ***L***atent ***D***irichlet ***A***nalysis (**LDA**) is the Bayesian version of pLSA. It categorizes documents, treated as Bag of Words, by topic via a generative probabilistic model assuming they are produced from a mixture of topics. Dirichlets distributions is better in finding the assignment of documents to a general topic.

In a corpus of M documents, it is necessary to discover k topics. LDA outputs the topic model and the M documents expressed a combination of topics finding the weight of connections between documents and topics and words. The algorithm creates an intermediate layer with topics and finds the weights. In this case documents are no longer connected to words but to topics.

A Dirichlet distribution $Dir(p)$ is a probability function which gives probabilities for discrete random variables. It includes the concentration parameter p that rules the trend of the distribution:

- Sparse, producing a real life distribution: $p < 1$;
- Uniform, producing a random distribution: $p = 1$;
- Concentrated: $p > 1$.



In LM there are two probability functions:

- $\theta_d \approx Dir(\alpha)$, the probability of topic k occurring in document d;
- $\psi_k \approx Dir(\beta)$, the probability of word w occurring in topic k.

The main difference between LSA and LSA is the assumption of the Dirichlet distribution, while the second does not assume any distribution. LSA works better than pLSA because it can better generalize new documents.

7.3 Evaluation

There are different approaches evaluating topic modeling:

- Eye Balling Models, identifying the top n words in a documents;
- Intrinsic Evaluation Metrics, using the *Perplexity*, a measure of uncertainty, and the *Coherence*, measuring the semantic similarity between top words within the topic. They help to distinguish topics in an interpretable topics from topics that are artifacts of statistical inference;
- Human Judgments, so what is a topic;
- Extrinsic Evaluation Metrics, so if the model is good at performing task, such as classification.

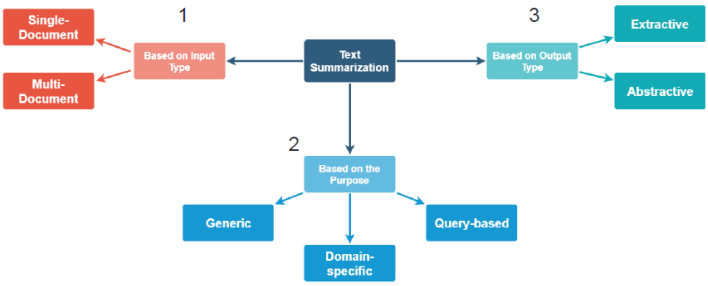
8 Topic Classification

Topic Classification is a supervised Machine Learning technique in which there are a list of predefined topic for a set of texts. There are many approaches:

- Rule-based Systems, programming a set of hand made rules based on the content of the documents
- Machine Learning Systems, like ML;
- Hybrid Systems, a mixture of the previous two.

9 Text Summarization

Text Summarization is the process of finding the most important information from a source to produce an abridged version of a text for a user. Main advantages are the reduced reading time and the easy research of a document. Automatic summarization improves the effectiveness of indexing. There are different dimension of the Text Summarization:



The Text Summarization *Based on Input Type* is the process of summarizing the text, given a single document (or a group of documents). Given a multiple document, it is possible to produce a gist of the content (E.g: series of news stories on the same event).

The Text Summarization *Based on the Purposes* can be: * Generic, with no assumptions about the content of the text to be summarized (a generic summarization); * Domain-Specific, using a domain knowledge forming a more accurate summary; * Query-Based, summarizes a document to an information need expressed in a query. Sometimes there is also a snippet summarizing a Web page.

The Text Summarization *Based on Output Type* can be *Extractive*, which selects some important phrases from the input text, or *Abstractive*, which captures the meaning of the text and producing a summary based on its own phrases (more appealing).

The Extractive Summarization has 3 different tasks:

1. Create an intermediate representation of the input text capturing the key aspects. There two main approaches:

    - *Topic Representation*, an intermediate representation capturing topic in the input text and sentences are scored for the importance . It is possible to use topic words (table of words and corresponding weights), LSA or LDA. The importance of a phrase is computed as the number and proportion of topic signature in the sentence. When assigning weight it is possible to use Word Probability or Tf-Idf. It is possible to use also the SumBasic system to determine the sentence importance. For each sentence in the input, it assign a weight equal to the average probability of the words in the sentence:

    $$g(s_j) = \frac{\sum_{w_i \in s_j} \mathbb{P}(w_i)}{|w_i | w_i \in s_j|}$$

    The algorithm selects the best scoring sentence containing the highest probability word. After this, the probability of each word appeared is adjusted (for example, the probability of a word occurring in a summary is lower than a word occurring one time).
    - *Indicator Representation*, where the text is represented by different set of possible indicators of importance. It is possible to represent each sentence as a list of indicators of importance (E.g. sentence length and location in the document). Graph-based methods and Machine Learning techniques are employed to determine the important sentences included in the summary;

2. Scoring sentences based on the representation, where each sentence is assigned to a score indicating its importance. The weights of each sentence is determined combining different indicators using Machine Learning techniques;
3. Selecting a summary consisting of several scored sentences. It is possible to show the best n approaches or the maximal marginal relevance approaches;

9.1 Evaluation

It is difficult to evaluate a summary because there isn't an ideal summary for a document. Only human summarizers have low agreement for evaluating and producing summaries. There are also metrics to automatically evaluate summaries. One of the most important is the *ROUGE* metric, evaluating the quality of the summary comparing it to human summaries. There are several variation of ROUGE:

- ROUGE-n, based on comparison of n-gram of the candidate summary and the other extracted from the reference summary only;
- ROUGE-L, based on the Longest Common Subsequence between two sequences. The longer LCS between two summary sentence, the more similar they are;
- ROUGE-S, allowing insertions of words between the first and last words of the bi-grams (they don't need to be consecutive sequences of words).

10 Information Retrieval

Information Retrieval is an old problem in computer science: its main purpose is to find a documents on the Web. The main problem is the fact that documents are distributed between servers and not stored on the same computer; but also the quantity of data (and how to reach as many pages as possible) that are constantly updating can be very difficult to manage. Documents are managed in different ways, according to various criteria. Information Retrieval is done by *search engines*.

It is important to understand user's needs, to interpret the context and to yield documents that answer the query, according to its relevance. First, Search Engines considered only topic relevance: it was not important to interpret a user query but just to yield relevance documents. Now the idea is to give a better user experience: the search process considers other variables such as geographical location and previous query done by the user; it can also be done without a user query (*Information Filtering Systems*) just filtering a stream of documents in real time.

In a search engine the query is usually not written in a formal language (but in a natural language): conditions have to be extrapolated from the text to reduce number of results. Information can be extrapolated from supports of different kinds: there are a lot of technical (due to different formats) and semantic (information is synthesized and represented preserving its meaning) problems.

A search engine is composed by two aspects (off-line and on-line): in the first part (offline) documents have to be indexed (and updated regularly by sub-parts of the archive) and represented in a formal way so that query

can be done; than in the online part, the query is parsed to represent it in a logical language and than the match is made, according to classic set theory (binary matching) or using a similarity index on a vector space (cosine similarity, proximity and so on, that permit a fuzzy match).

10.1 Indexing

Documents have to be indexed to retreive them easily: this process extracts a small portion of text that will be used to search a document. This process is extremely resource expensive, so are implemented in a particular way to optimize the process. Each document is characterized by a different set of terms weighted in a different way: documents are indexed to reduce the dimensionality of the matrix so that the number of terms will be lower than the number of documents. The resulting matrix is inverted so that documents are on columns and terms on rows: retrieving a file is just as easy as checking in which document the term is present. The data structure used in practise is a dictionary in which the key is the term, the value is a list of tuples `<document; weight>` so that looking for a document will not scan useless documents.

There are cases in which index can not be stored on RAM due to amount of documents: in this case the use of a column-storage database system is required. In fact this strategy garantees both efficent management of a sparse matrix and scalability on a cluster of machines.

10.2 Matching

The query the user ask to the search engine is represented in a formal way to compare it with the (indexed) database. To speed up the process, the database is inverted (inverted-index) so that documents are indexed basing on terms, ordered by relevance (if available) or by progressive number (if the scoring is abstent or done *on-the-fly*).

10.2.1 Boolean matching

Boolean matching is based on set theory:

$$R(d_j) := \{t_i | w_{ij} = 1\} \qquad w_{ij} \in \{0, 1\}$$

the document is a set of terms, so relevance is modeled as a binary property (a document is either relevant or not, ranking is not permitted). A query is formally represented by boolean operations like `AND`, `OR` or `NOT`. Using the inverted representation the matching process returns a sub-set of documents that satisfy user's query. Evaluations order of operations must be clearly specified because a different order can produce different results. In general operations are a priority order pre-imposed in the representation process.

The evaluation is made using a recursive algorithm on a binary tree (the formal representation of the query) starting from leaves, with some optimizations that evaluate longer list for lasts (to save memory).

This matching model is fast and simple, but it is not able to produce a ranking of results: in some cases it is not sufficient for accomplish the task.

10.2.2 Vector Space matching

To solve problems of the Boolean model, the Vector Space model is presented: documents are presented as vectors (defined by a word-embedding algorithm) in a space with a number of dimensions equal to vocabulary size $|V|$. Documents are represented using some indices (like presence-absence of the term, (log or normalized) frequency, tf, $tf\text{-}idf$ value…) as vectors (points) in this space. In general, are used:

$$w_{t,d} = \frac{tf_{t,d}}{\max\limits_{t_i \in d} tf_{t_i,d}}$$

$$w_{t,d} = \frac{tf_{t,d}}{|d|}$$

$$w_{t,d} = \begin{cases} 1 + \log(tf_{t,d}) & tf_{t,d} > 0 \\ 0 & \text{otherwise} \end{cases}$$

Also the query is presented as a vector, so that the matching problem is just as easy as compute similarity between vectors or using a KDTree to cache documents.

11 Web Search

As its name says, the Web is structured as a graph, dynamic and oriented. Edges reachable in both direction ($A \to B$ e $B \to A$) are told *Strong connected spaces*: the whole structure is often abstracted as a set of Strong Connected Spaces instead of pages. In this way the graph assumes a *papillon* structure: there is a *source component* (from which it is easy to reach other pages but difficult to access), a *huge component* (that represent the majority of documents) and a *shaft component* (easily reachable but from which it is difficult to exit); there are also some noisy documents isolated or random linked. To retrieve a document, a web search engine must

had indexed it: documents not retrievable by search engines belong to *deep web*. Those documents represent the majority of the web and include dynamic web pages (which content changes according to a input), private pages (hidden by CAPTCHAs or login) or just because there are no incoming links.

Web search can be done by browsing (using links between pages), direct link (if the address is known) or using a Web Search Engine. First generation of Web Search Engine considered only documents word frequency, but then also links are considered to rank documents by authority. *Semantic Web* considers also the meaning of the documents to improve user experience.

11.1 Updating documents: Gathering

Gathering is the process of index updating by a search engine: due to mutability of documents in the web, index have to be constantly updated to include new documents or to update existing ones. This process can be automatically done asking the search engine to update the document manually, but is ofen done in an automatic way using *crawlers*: bots that surf the net following links and storing documents. A *crawler* starts its search process with a set seed of reliable pages (such as public organization) taken randomly; then it checks the presence of unseen links in the text, stores them and opens one (per thread) so that a server is not filled of requests at the same time (latency is given by *politeness rules* in the `robot.txt` file). Then the document is processed and usfull information (such as url, title and content) are stored in a column-based database. The process goes on until there are no more links to visit or the disk is full.

During the process, also metadata are stored, according to politiness rules (written on a sitemap file called `robots.txt`): those meta-data include importance, update log and frequency for each page, latency of requests and so on. Documents are then cleaned to parse text: it is considered

11.2 Ranking

Pages are retrieved by a score calculated on relevance to the query, popularity and position in the Web structure. The main idea is that a good document (a document with high authority) is linked to many other good documents (*Hub Score*: analyzing the Web graph it is possible to estimate a document authority to improve the search.

11.2.1 Pagerank

It is Google algorithm: the main idea is to simulate a *Random walk* across the web: a user, every fixed time, goes to a random page on the web (with a very low probability of λ) or follows a link in the current document. The score given by *PageRank* is the probability that in a random moment the user is reading that page.

$$PR(p) = Kd + K(1-d)\sum \frac{PR(p_i)}{C(p_i)}$$

where K is a normalization constant, d depends on the system and $C(p)$ is the number of documents of the page p. The fact that PR is in both side of the formula means that it is computed recoursively until convergence.

11.2.2 HITS

This algorithm estimates both authority and Hub Scores for each page: the process is iterative and converges after a small number of passages (depending on graph size); during estimation only graph structure is considered. The algorithm initialize each document with both scores equal to 1 and for each iteration updates them according to formulas:

$$A(p) = \sum_{q \to p} H(q)$$
$$H(p) = \sum_{q \to p} A(q)$$

12 Recommender Systems

A *Recommender Systems* is a push technology: it provides the user with contents without a query; it suggests other contents according to user's behaviour and previous queries. A good recommender system should not suggest common objects but unknown items that users do not know but might like. The system should consider user's feedback and not only contents that he or she had actually searched: a relevance score is extracted during the retrieval process. In general the systems are based on *Information Filtering*: contents are monitored producing data (virtually in real-time) that have to be filtered to inform the user.

A recommender system can follow different paradigms, and there are also hybrid approaches. Assumptions of a recommender system are that users rate and catalog items and that a person will not change tastes during time. This second assumption is too strong for a real

environment: techinques were developed to update a user profile making the process dynamic (computing users embeddings in a never-ending learning pipeline).

12.1 Collaborative filtering

In this architecture the user is clusterized and contents are suggested according to the belonging cluster (a not-seen content is presented if is appreciated in the cluster). This approach is based to data provided by users: to get more data, some indicators are stored (such as web-surfing behaviour) to improve the process, although the behaviour can be misunderstood.

At beginning there are no user information: so he or she is forced to rate a set of items, demographic data are used to do some assumptions or a default set is presented.

In this paradigms, an important part is to compute the *User-item Matrix*: the output is a numerical prediction that indicates if user will like a certain item or not. So a top N list can be extracted. That matrix is a sparse matrix with users stored as rows and items as columns.

Given a *target user* a, an item i not yet seen by a is suggested after that procedure:

- Is extracted a subset of users $U_{a,i}$ similar to a that have reted item i;
- A model (machine learning or an heuristics) is provided to compute a rating $\hat{r}_{a,i}$ for item i according to $U_{a,i}$;
- If $\hat{r}_{a,i}$ is greater a certain value, i is suggested to a.

The subset $U_{a,i}$ is extracted computing the Pearson correlation ϱ between ratings of a and each other user $U_{a,i} = \{b | \varrho_{a,b} \geq r \ \forall b \in \text{Users}\}$. A common predicting function for $\hat{r}_{a,i}$ is:

$$\hat{r}(a,i) = \overline{r}_a + \frac{\sum_{b \in U_{a,i}} sim(a,b) \cdot (r_{b,i} - \overline{r}_b)}{\sum_{b \in U_{a,i}} sim(a,b)}$$

This process have scalability issues: number of users is much higher than number of items $U >> I$, so computing similarity between users can be very expensive. In addiction, operating with very sparse matrices, is very difficult to have a match between users.

To respond to those problems, other approaches are used: for example, an item-item matrix is computed, so that the prediction $\hat{r}_{a,i}$ is computed with the formula:

$$sim(\vec{a}, \vec{b}) = \frac{\sum_{u \in U}(r_{u,a} - \overline{r}_u)(r_{u,b} - \overline{r}_u)}{\sqrt{\sum_{u \in U}(r_{u,a} - \overline{r}_u)^2} \cdot \sqrt{\sum_{u \in U}(r_{u,b} - \overline{r}_u)^2}}$$

$$\hat{r}_{a,i} = \frac{\sum_{p \in R_u} sim(\vec{p}, \vec{i}) \cdot r_{u,p}}{\sum_{p \in R_u} sim(\vec{p}, \vec{i})}$$

where R_u is the set of items rated by the user u.

A machine learning model (trained on a subset of users) can be used to predict future ratings; this approach requires regular updating of the model.

12.2 Content-based filtering

The algorithm considers only features of all the items and the user profile (with contextual parameters): a similarity measure is than used to match the profile with items. This approach is not based on a community, so it works even with a few users. The content can be presented in a structured or unstructured way, and is used to compute the Cosine Similarity $sim(\vec{i}, \vec{u})$ between user behaviour and item.

Another simpler approach is to use a Nearest Neighbors approach: given a set I_a of items rated by the user a, the k not-yet-seen nearest items $i \in (I - I_a)$ are presented to the user (according to cosine similarity). Users' rating can be also used as additional information to improve rating performance.

12.3 Knowledge-based filtering

This approach is similar to the previous one but products are stored in a knowledge base that should improve the item embedding and filtering.