Text Mining and Search

Alberto Filosa

29/9/2020

Indice

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` $\rightarrow$ `be`

`car`, `cars`, `car's`, `cars'` $\rightarrow$ `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` $\rightarrow$ `automat`

3 Text Representation

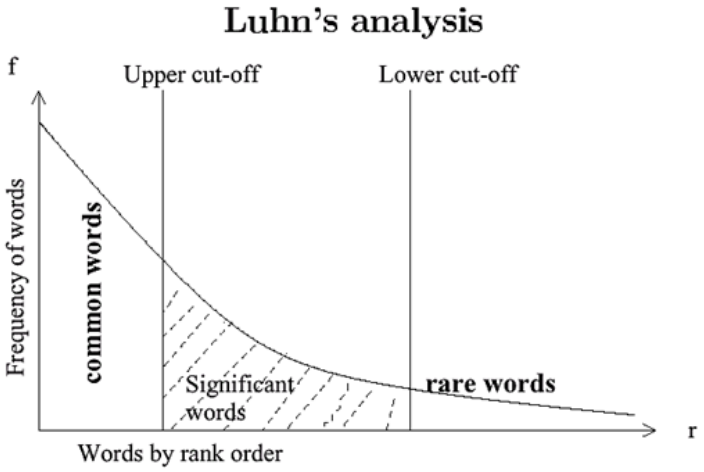
Text Representation is the process to representing the text with graphical method. The simplest way is the binary term-document weighting, in which each document can be represented by a set of term (or binary vector) $\in \{0, 1\}$. Another graphical method is the count matrix,

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.

4 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: iaeQ45YB@sYwv&pell Correction;

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

The main task is to compute the probability of a sentence (or a sequence of words), $\mathbb{P}(W) = \mathbb{P}(w_1, w_2, \dots, w_n)$, while the related task is 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:

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

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 LM 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):

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

4.1 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 y 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 and apax 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 $\alpha = 0.75$):

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

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*.

4.1.1 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.

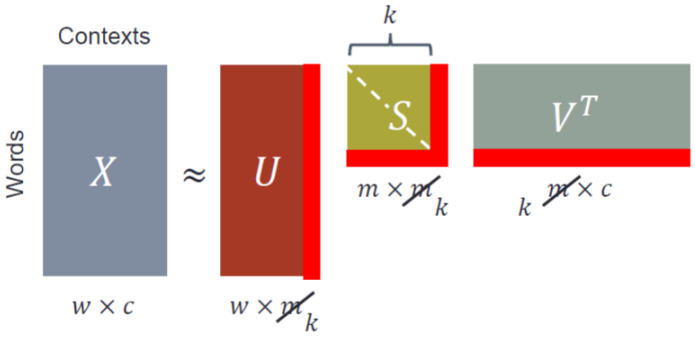
The most popular model is the ***S**ingular **V**alue **D**ecomposition* (**SVD**), a dimensionality reduction techniques used in Text Mining. Any rectangular $w \times c$ matrix X can be expressed as the product of 3 matrices:

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

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.

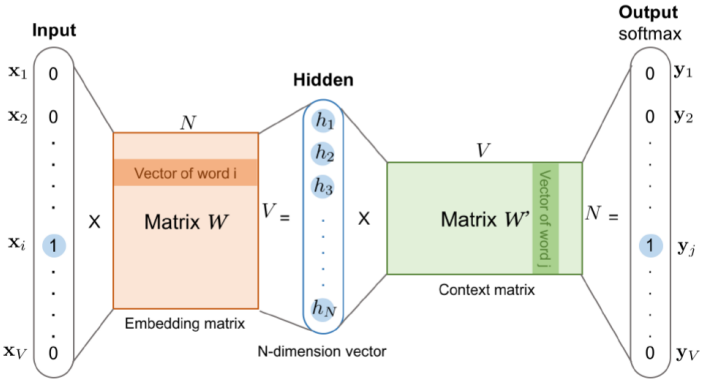
It is similar to Language Modelling, but it predicts the context rather than next word. The loss function is the following:

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

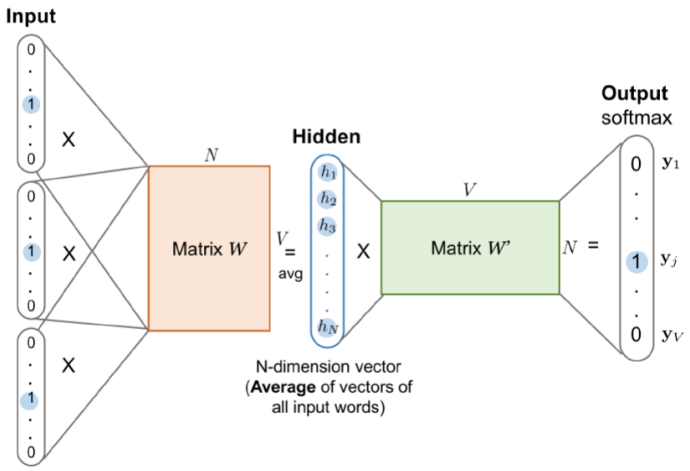
There are two training method:

- 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 to which data items belongs to a predefined finite set of classes. In text classification, data items can be textual (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 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.

5.1 Applications

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.

5.2 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:

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

It can be used also with matrix decomposition like SVD, PCA, aggregating different features.

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.