Clustering of short sentences through representation of text data

MAXIME HOUÉ
Abstract

Natural Language Processing has developed in the past few years very quickly. Numerous new applications emerged from new methods, notably involved by the creation of the popular word embedding Word2Vec created by a team of Google researchers. One of these new applications is the chatbot technology. The goal of these conversational interfaces is to be able to communicate automatically with humans via written or voice chat. With a chatbot, a company hopes to improve customer relations at a lower cost. Unfortunately, the skills of the chatbots can vary a lot, and until now, their understanding of the humans is often rather bad. This harsh conclusion leads to wonder how the chatbot developers can be helped for handling the large amounts of user requests not understood by their chatbot.

This thesis was made in collaboration with a start-up named Askhub. This start-up aims to help the companies with the development of their chatbot.

The aim of this master thesis is to propose a clustering system in order to classify the data not understood by a chatbot. To begin with, a study of the different methods of word embeddings has been realized, followed by a study of different clustering techniques available suitable to the chosen word embedding. The results are then compared with some metrics and some propositions were made in order to improve the clustering results.
Sammanfattning


Detta examensarbete gjordes i samarbete med en start-up som heter Askhub. Denna uppstart syftar till att hjälpa företagen att utveckla sin chatbot.

Syftet med denna detta examensarbete är att föreslå ett klustringssystem för att klassificera data som inte förstås av en chatbot. Till att börja med har en studie av de olika metoderna för word embeddings gjorts, följd av en studie av olika klusteranalyser som är lämpliga för det valda word embedding. Resultaten jämförs sedan med vissa mätvärden och några förslag gjordes för att förbättra klusteranalysresultatet.
Acknowledgment

This degree project for the master’s degree in Information and Network Engineering was conducted at the start-up Askhub in Paris.

I am thankful to Hatim Khouzaimi for being my supervisor at Askhub. I am also grateful to my supervisor Saikat Chatterjee at KTH Royal Institute of Technology.

I would like to give my last acknowledgements to all my co-workers at Askhub.
# Contents

1 Introduction 1

1.1 Context 1

1.1.1 Natural Language Processing 1

1.1.2 Chatbot industry 2

1.1.3 Adequation with the Sustainable Development Goals 3

1.2 Thesis project 3

2 Global model 4

3 Cleaning of the text data 5

3.1 Initial cleaning 5

3.2 Stopwords 5

3.3 Spellchecker 6

3.3.1 Type of errors 6

3.3.2 Spellchecker processing 6

4 Vectorization of text data 8

4.1 Word embeddings 8

4.1.1 Word2Vec 8

4.1.2 GloVe 10

4.1.3 FastText 12

4.1.4 Semantic knowledge graph 13

4.1.5 Relationship between the models 19

4.2 Sentence embeddings 19

4.2.1 Mean embeddings 19

4.2.2 Word embeddings with TF-IDF 19
5 Clustering
  5.1 Clustering algorithms
    5.1.1 K-means Algorithm
    5.1.2 Minibatch K-means
    5.1.3 Graph linkage clustering
    5.1.4 Gaussian Mixture Models
    5.1.5 DBSCAN
  5.2 Relevance of a clustering on text data
    5.2.1 The silhouette coefficient
    5.2.2 Calinski-Harabaz Index / the variance Ratio Criterion
    5.2.3 Davies-Bouldin Index

6 Implementation
  6.1 Working framework
    6.1.1 Input data
    6.1.2 Cleaning
    6.1.3 Choice of the word embeddings
    6.1.4 Clustering methods
  6.2 Results
    6.2.1 Identification of the number of clusters through K-means clustering
    6.2.2 K-means
    6.2.3 Gaussian Mixture Models
    6.2.4 DBSCAN
  6.3 Comparisons between the results

7 Conclusions and Future Work
  7.1 Conclusions
    7.1.1 Word embeddings
    7.1.2 Clustering algorithms
  7.2 Future Work
List of Figures

2.1 Scheme of the processing of the text data .................................. 4

3.1 Scheme of the cleaning phase ................................................... 5

4.1 Example of the Skip-Gram model .............................................. 9
4.2 Example of the Continuous Bag Of Words ............................... 10
4.3 Illustration of the node "ConceptNet" taken from the website  conceptnet.io ............................................................. 17

5.1 One limit of the K-means algorithm ........................................ 23

6.1 Elbow function obtained through K-means minibatch .................. 34
6.2 Silhouette score for K-means algorithm with GloVe embedding .... 35
6.3 Silhouette score for K-means algorithm with ConceptNet Numberbatch embedding .................................................. 35
6.4 Silhouette score for Gaussian Mixture Models algorithm with  GloVe embedding ......................................................... 36
6.5 Silhouette score for Gaussian Mixture Models algorithm with  ConceptNet Numberbatch embedding .............................. 36
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.1</td>
<td>The co-occurrence matrix for the example sentence: &quot;The cars were racing in the middle of the night.&quot;</td>
<td>11</td>
</tr>
<tr>
<td>4.2</td>
<td>The score table of the different instances for the concept &quot;apple&quot;</td>
<td>14</td>
</tr>
<tr>
<td>4.3</td>
<td>The different Hearst patterns</td>
<td>14</td>
</tr>
<tr>
<td>4.4</td>
<td>Sizes of the different core languages of ConceptNet</td>
<td>17</td>
</tr>
<tr>
<td>6.1</td>
<td>Results of the algorithm with the different scores</td>
<td>37</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Context

1.1.1 Natural Language Processing

Conceptually, natural language is a way to communicate our thoughts to other persons. In other words, our thoughts are encoded into a language, and then the language is decoded by other people. Natural language is ubiquitous in the human society. When someone is talking to another, natural language is implied. When a person is writing a book, he is encoding the thoughts he had and the future readers of this book will decode all data written in it. This encoding and decoding of language is not specific to humans.

Many animals have their ways to communicate with each other. Whales for instance use their vocalization to communicate to each other. Even bees are flying in a certain way when they want to share some information with their congeners. Indeed, they are performing a dance, called the waggle dance to indicate the direction and the distance between where they are located and the flowers to forage. Communicate from each other in the animal world is very common.

However, human languages are far more complex to encode and decode afterwards. Sometimes, a new concept appears and it is not in the language vocabulary. In this case, humans create new words, terms or expressions to define this new idea. Some of the biggest particularities of language, and why it is so hard to describe and comprehend for machine is polysemy and context. Many words will have different meanings according to the sentence and its context around it. When two people are communicating to each other, they are using their common knowledge to understand each other. This common knowledge needed for understanding human language is something really hard to grasp for computers. A simple sentence like this "A bug has found its way into the computer." can create big issues for machines. The obvious meaning for us humans is that something is wrong in the computer.
and that some programs are not working properly. A machine may understand the sentence literally and connect the word "bug" to the word "insect". Thus, the sentence would not make much sense by meaning that an insect snuck into the hardware of the computer. Even though it seems pretty straightforward as something innate for humans, modeling the natural language for computers is still a large and hard task in progress.

Natural Language Processing (NLP) aims to generate automatic analysis and depiction of human language with the assistance of theory and computational techniques. At the beginning of NLP research, computations were very time-consuming but a lot of techniques are now instantaneous. The data stored on the Internet has increased quickly and has been a very valuable resource for Natural Language Processing. A large part of Internet data is unstructured, mainly due to the fact that the Internet data was not intended for computers but for humans. Thus, analyzing text for a computer was a hard task and still is now.

First algorithms were very performant at computing word frequencies or detecting misspelled words, mostly due to the fact that it was the most straightforward and feasible thing to do at that time. However, these algorithms did not catch any semantic in text. Still today, many algorithms are based on this, while the limitations of such processing are numerous. They are only relying on the positions of the words in a sentence while a human is seeing different concepts behind every word.

In 2013, an important breakthrough has been made in the domain of the Natural Language Processing with the arrival of the performant word embedding Word2Vec [1]. A word embedding is a technique for modeling the language. The Word2Vec technique links a vector for each word it has been trained for. With this new technique, many new applications have appeared in our daily life. One of these applications is the chatbot.

### 1.1.2 Chatbot industry

A chatbot is a program designed to converse with humans about a specific topic through voice or chat messages. There are many applications for such technology for a company. For instance, it can be used to improve the relationship between a company and the visitors of its website by helping them to navigate through it. This is one of the reason why chatbots are often thought to behave the most possible like an human in order to be the most appealing for users. With more and more people using messaging applications, the chatbot industry has a bright future ahead.

However, the chatbots are far from understanding everything the humans tell them. Therefore, the chatbot developers need to know the defects of their chatbot, i.e. what the chatbot did not understand. The subjects occurring in these unparsed requests of a bot are often related to the field of action of the company. A chatbot can generate thousands of requests unparsed daily, and this is impossible in this case to process them all manually.
1.1.3 Adequation with the Sustainable Development Goals

This thesis is in line with some of the Sustainable Development Goals set by the United Nations.

The chatbot technology can contribute to improve the wellness of people. Indeed, there are chatbots designed to help ill people by providing them with advice and recommendations. Thus, the thesis is in adequation with the third goal of the Sustainable Development Goals about "Good Health and Well-Being".

By using very recent and innovative technology, this thesis is also matching the ninth goal of the UN Sustainable Developments Goals about "Industry, Innovation and Infrastructure".

1.2 Thesis project

Based on this observation, the start-up Askhub aims to help the chatbot developers through a SAAS (Software As A Service). The thesis focuses on implementing and evaluating different kinds of clustering on short texts. Due to the confidentiality of the activities of the start-up, the studies in this report have not been made on the original datasets, but on a dataset which was approaching the same properties.

The remainder of the report is structured as follows. The global model is proposed in Chapter 2. In Chapter 3, the cleaning phase is explained. Different techniques of word embedding are proposed in Chapter 4. In Chapter 5, a study about different clusterings has been made. The results are included in Chapter 6. Finally, conclusions and future work are discussed in Chapter 7.
Chapter 2

Global model

The overall procedure for clustering text data is described in the figure 2.1.

Figure 2.1: Scheme of the processing of the text data

The text data is firstly cleaned from noise through the cleaning step. Secondly, every request is vectorized with the help of a word embedding. Finally, the clustering is performed over the vectors previously generated.

To that, the step of evaluating the clusters can be added at the end of the process. Through this step, it will be possible to choose the best solutions.
Chapter 3

Cleaning of the text data

The phase of cleaning can be described as follows in the figure 3.1.

![Figure 3.1: Scheme of the cleaning phase](image)

3.1 Initial cleaning

At first, all the punctuation and the symbols are removed from the data. This is due to the fact that the word embedding will not be able to provide a vector to them. Furthermore, punctuation or symbols can interfere with real words. If not removed, the word "thesis?" will not be understood as the word "thesis" for example.

3.2 Stopwords

The next step concerns the stopwords. The concept was introduced by P. Luhn, H [2]. A stopword is a frequent word that does not bring any valuable new information to the actual sentence or text. Therefore, those words would be omitted in the cleaning phase. Typical stopwords are the determinants like "a" or "the". In some case, the stopwords may add noise at the vectorization step.

The fact of removing the stopwords is debatable. Even though those words are not bringing any information on a quite long text, they may have a greater influence on shorter texts or sentences. In the case of a chatbot user, there is a significant chance that the request is constituted only by stopwords. If it is happening, the whole request is discarded and will not be visible in the clustering step.
3.3 Spellchecker

In this section, we will tackle the problem of spellchecking. The goal here is to reduce the number of words not understood by our model. This step can be important since the model does not assign a vector to words that are not in the vocabulary. A first implementation of spellchecker is described in the paper [3].

3.3.1 Type of errors

The spellchecker’s aim is to correct misspelled words in a text. One common example of its use is to highlight misspelled words in a word processing tool or to correct a typo when you are writing a message on your cell phone. A spellchecker can correct different sorts of spelling errors. The most common error is the non-word errors. The misspelled character has created a word which does not exist in the dictionary. For instance, typing "thesi" instead of "thesis". The second type of errors is the real-word error. This time, the error has generated a word which exists. These errors are harder to correct than the non-word errors. For instance, if someone is typing "hand" instead of "and". The third type is the cognitive error and happens when someone is confusing the orthography of a word. Contrary to the first types of errors, these errors are not due to an error of typing but to an error of the person. For instance, this error happens for words with same utterance but different orthography.

3.3.2 Spellchecker processing

We search to determine what is the intended word $w$ from the typed word $\hat{w}$. For instance, if the typed word $\hat{w}$ is very common, like “have”, the intended word $w$ is very likely to be the written word $\hat{w}$, “have” here. If the written word $\hat{w}$ does not exist, like the word “jave”, the word which is the closest in terms of string distance and the most common should be the intended word $w$, the word “have” in our case. However, if the typed word exists in the language but it is quite uncommon in the same time, it is harder to tell which one is the best.

More generally, a set of candidate words $I(\hat{w})$ is created from the written word $\hat{w}$. For instance, for the word $\hat{w} = \text{“shave”}$, the set $I(\hat{w})$ is [“have”, “share”, “shame”, ...].

The set of candidates $I$ is created by using the distance of Damerau - Levenshtein. This distance is measured between two strings and is simply the minimum number of operations needed to switch from one word to another. These operations include insertions of new characters, suppressions of old ones, replacement of old characters by new ones or inversion between two old characters. Usually, the candidate words are created by taking all possible candidates with a distance of Damerau-Levenshtein equal to one, because a small mistake is more likely to happen than a big one. However, the chosen distance can be higher but it will impact the performances of the model since it will increase the number of elements in the candidate set. The number of elements in the candidate set is already significant with a distance of one. For a word length of $L$ and a number of characters in the alphabet equal to $n_c$, we can do $n_c (L + 1)$.
insertions, $L$ suppressions, $n_c L$ replacements and $L - 1$ inversions leading to more than $(2L - 1)(n_c + 1)$ candidates. However, among all candidates, only a fraction of them are real words so it will restrain a lot the size of the set $I$.

When the set of candidates is done, the goal is to maximize the probability $P(w|\hat{w})$ for $w$ in the set of candidate words $I$:

$$\arg\max_{w \in I(\hat{w})} P(w|\hat{w}) = \arg\max_{w \in I(\hat{w})} P(\hat{w}|w)P(w)$$ (3.1)

A simplistic model would consider that $P(\hat{w}|w)$ is constant for any word $w$ proposed in the candidate set. In this case, finding the best candidate will come to find the most frequent word of the set of candidates. Those frequencies can be easily retrieved from a large corpus of text of a given language.

A more complex model could be trained from an annotated dataset in order to estimate the probability $P(\hat{w}|w)$. The annotated dataset is describing the most frequent spelling mistakes with the correct form next to it. It could also take into consideration the positions of the letters on a keyboard. Indeed, the positions of the keys are playing a major role in the misspelling errors.
Chapter 4

Vectorization of text data

The second step of our model is the vectorization of the cleaned text data.

4.1 Word embeddings

A word embedding is a method in which each word of a vocabulary is mapped to a real vector. The interest of word embedding is that the data from natural languages is discrete. There are only words composed of characters. Those characters are defined arbitrary and do not give any useful information. For example, we take two words with a very close meaning like "city" and "town". If we are only looking to the characters of each word, we will not be able to say that the two words have a very close meaning. However with a word embedding, we can compute a distance between the two vectors representing the two words and this distance will be quite low with a good word embedding. There are many situations in Natural Language Processing where a word embedding is very useful.

The modern word embeddings can show some very interesting properties, notably in terms of linearity between some words. For instance, the vectors of some words are relatively close to each other. The closest word vector of the vector which equals the vector of "Madrid" minus the vector of "Spain" plus the vector of "France" is the vector of "Paris".

One of the first implementations of such models like word representations dates back in 1986 with the work Rumelhart, Hinton and Williams [4]. Their ideas have been developed since and have revealed being very efficient for designing applications such as speech recognition.

4.1.1 Word2Vec

The most famous models used to generate word embeddings are Word2Vec, which was created by Tomas Mikolov and his team at Google [1]. The words representations computed with Word2Vec show some very valuable properties.
One of the very interesting features found is the linearity between the vectors of some “close” words.

The Word2Vec is implemented with a neural network. This neural network includes an input layer, a hidden layer and an output layer. However, the interest is not in the output of this neural network. The results for what the model was trained will not be used. The whole consideration will be given to the weights of the hidden layer. Those weights are considered as word vectors. Word2Vec can use two methods for generating word embeddings. It can use the Skip-Gram Model or the Continuous Bag of Words (CBOW). To explain a bit further how the models work, we will take the following sentence example : “It is raining cats and dogs.”.

The first method is trying to predict the context of a word base on this word. If you are focused on the word “cats” and you have a window size equal to two, you would like to predict the surrounding words of “cats”, i.e. “is”, “raining”, “and” and “dogs” [4.1].

![Figure 4.1: Example of the Skip-Gram model](image)

Reverse, the second method is used to predict the target words based on the context words. Therefore, it means that with the context words “is”, “raining”, “and” and “dogs”, we would like to guess the word “cat” with a window size equal to 2 [4.2].

Initially, every single word in a Word2Vec owns one dimension of the generated space. It means that for each word $w$ of the vocabulary, the vector $v_w$ associated to the word is a one-hot vector and that the generated space has a dimension equal to the size of the vocabulary. As a space dimension equal to the size of the vocabulary would be too unpractical for computers to deal with, the dimension of the space is lowered to a number between 100 and 400.

Given a word in a sentence, we will take a random word which is a neighbour of the first word. This neighbour must be inside the window defined in the algorithm. A typical value for this window size is about 5 words, 5 words behind and after the target word. Thus, we can compute the probability for every word within the vocabulary of being a neighbour of our target word. For instance, after we have done the training and if we are looking to the word “united”, the probabilities for words “states” or “kingdom” will be higher than for words
Figure 4.2: Example of the Continuous Bag Of Words

which do not have any connection with it, like “chocolate” or “thesis”.

One of the hyperparameters is the number of features of the vectors and corresponds to the number of neurons in the hidden layer.

The advantage of Word2Vec is that it is an unsupervised classification. Thus, it is not relying on text data being annotated, which was the biggest breakthrough with this work. Large amounts of annotated data is very hard to gather in Natural Language Processing.

Word2Vec has however some limits. One of the problems encountered with a simple word representations is the absence of representations for composed words or concepts like "New York" or "electronic music".

Many Word2Vec dictionaries are available in several languages and have been trained on large corpus of data.

4.1.2 GloVe

One of the drawbacks of the Word2Vec solution to generate word embeddings is that Word2Vec is not taking in account the statistic of occurrence of a word in a large corpus. Actually, Word2Vec is only considering the local context, according to the size of the previously predefined window. The Global Vectors for Word Representation [5] algorithm main motivation is to consider these statistics and global counts when generating word embeddings. The first idea for generating word vectors already used the concept of global statistics upon every words in a large corpus of text. However, one of the very interesting features present in the embeddings generated by Word2Vec was missing : the linearity between some of the word vectors as described previously.

This property is something the researchers wanted to have in the model because it is showing that the model seizes a true meaning and a true connections between different words. The objective of GloVe is to take in account the global statistics of words while at the same time keeping the linearity relationships.

The approach for GloVe algorithm is to build a very large co-occurrence
matrix. The context for words is also captured with a window size. For instance, the co-occurrence is the table 4.1 if we take a window size equal to two and this sentence "The cars were racing in the middle of the night.". As you can observe, the matrix is symmetric.

<table>
<thead>
<tr>
<th></th>
<th>the</th>
<th>cars</th>
<th>were</th>
<th>racing</th>
<th>in</th>
<th>middle</th>
<th>of</th>
<th>night</th>
</tr>
</thead>
<tbody>
<tr>
<td>the</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>cars</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>were</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>racing</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>in</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>middle</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>of</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>night</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.1: The co-occurrence matrix for the example sentence: "The cars were racing in the middle of the night."

The thing that GloVe is trying is the following statement: if two words often appear within the same context, their meanings are strongly correlated. Therefore, we can expect that some words like “forest” and “tree” will occur at the same level with the context “fire”. However, “tree” will occur more often with the context “fruit” than the word “forest”.

We will now use $X$ for denoting the co-occurrence matrix. $X_{ij}$ will represent the number of time the word $W_j$ has occurred with the context word $W_i$. $X_i$ will denote the number of words that are present in the context of the word $W_i$. In the following, the word vectors of the word $W$ will be denoted as $w$.

The previous example seems to indicate that the frequency of such word in the presence of such other word has a significant impact on the meaning of both words. Therefore, we will start the explanation on the following frequency $F$:

$$F(w_i, w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

(4.1)

$w_i$ and $w_j$ are denoting two word vectors, $\tilde{w}_k$ correspond to several context word vectors. $P_{ij}$ is the probability that the word $W_i$ appears next to specific context word $W_j$. As we explained before, one of the prerequisites behind GloVe implementation is to have a linearity between different vectors. In order to have this linearity, we express the function $F$ as follows:

$$F(w_i - w_j, \tilde{w}_k) = \frac{P_{ik}}{P_{jk}}$$

(4.2)

There are still some issues with this expression since we have vectors inside the function $F$ and a scalar value on the right side. Therefore, we will take the dot product between $w_i - w_j$ and $\tilde{w}_k$:
The expression that we have here does not allow to swap $w$ with $\tilde{w}$ which is problematic since we can swap a word with its context words without modifying the global count of the words. We also have seen that the matrix $X$ must be symmetric. The solution to this latter problem is this homomorphism:

$$ F((w_i - w_j)^T \tilde{w}_k) = \frac{F(w_i^T \tilde{w}_k)}{F(w_j^T \tilde{w}_k)} \quad (4.3) $$

The solution to the equation $F$ is $F = exp$. With the previous result, we obtain $F(w_i^T \tilde{w}_k) = P_{ik}$. With the equation $4.3$ we find that:

$$ w_i^T \tilde{w}_k = \log(P_{ik}) = \log(X_{ik}) - \log(X_i) \quad (4.5) $$

Hence:

$$ w_i^T \tilde{w}_k + b_i + \tilde{b}_k = \log(X_{ik}) \quad (4.6) $$

With this last relation, we can eventually define a least squares problem:

$$ J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w}_k + b_i + \tilde{b}_k - \log(X_{ik}))^2 \quad (4.7) $$

The weighting function solving this problem and chosen by the researchers was:

$$ f(x) = \begin{cases} 
(x/x_{max})^\alpha & x < x_{max} \\
1 & \text{otherwise} 
\end{cases} \quad (4.8) $$

According to the researchers, GloVe method generates word vectors which would surpass Word2Vec vectors in some situations.

The researchers at the origin of this embedding method have made available some dictionaries generated by following their method.

### 4.1.3 FastText

The previous models for learning and generating word vectors were considering the word as the fundamental unit. One of the downside of this approach is that those models did not take into account the different patterns within a word. Pattern refers here to the characters and their orders constituting the word.

The main idea behind FastText \[6\] is to represent each word as a list of n-gram which forms the whole word. With n-grams of size $n$ equal to 4 for the word “music”, we will have the following list: $[<mus, masi, usic, sic>]$. There is a vector for every existing n-gram used in the training data. The symbols “<” and “>” indicates respectively the beginning or the end of a word. The word vector $W_k$ is generated by the sum of all n-gram vectors $\sum_{g \in G_k} W_g$ with
\( G_k \) being the set of the n-grams of the word of index \( k \) and \( W_g \) the vector of the n-gram \( g \).

One of the advantages of FastText is that it can give better vectors for rare, out-of-corpus or new words. This can be very helpful and valuable in the event that someone made a typo in a word \( W \). Indeed, the word \( \tilde{W} \) will differ from the intended word \( W \) only of a character. Thus, the list of n-grams of the word \( \tilde{W} \) will be very close to the list of n-grams of the word \( W \). Actually, only two n-grams will be concerned by this typo. As the list of the n-grams will be almost the same, leading to two very close vectors.

### 4.1.4 Semantic knowledge graph

In this part, we will introduce the concept of knowledge graph and see how it can generate a word embedding useful to vectorize raw text data.

**Presentation**

A knowledge base is able to store any forms of data. The data can be structured or unstructured and is intended for computers. The data in a knowledge base is about a specific topic and comes from different and relevant sources. In our problem, we are more interested in structured data. A knowledge graph is a knowledge base with some kind of relations between the information.

We are interested here in semantic knowledge graph. Indeed, if we have a word, we would like to know the most related or close words. The aim of a knowledge graph is to help understanding as many words or expressions as possible. More generally, the data from the web is largely unstructured and thus hard to access for machines. Then, taxonomies aim to make all this data available and understandable for machines. There are several knowledge graphs available on the Internet: the Microsoft Concept Graph and ConceptNet. The main interests in these three knowledge bases is that the links are between word concepts. Thus, these knowledge bases can be used to enhance an existing word embedding.

**Microsoft Concept Graph**

The Microsoft Concept Graph is described in the research articles [7] and [8].

The current release of Microsoft Concept Graph involves only the relation “IsA” between a concept and an instance. This relation is implied between a “concept” and an “instance”, where the “concept” is an “instance”. More relations are planned in the future.

A score is also appearing in order to describe what the most probable instance is for a given concept. There are more than 5.4 million concepts, more than 12.5 million instances and around 87 million relations “IsA”. For instance, if the concept word is “apple”, the instance “fruit” will be at the top of the list, but the instance “company” will also be close to the top as you can see in the
Table 4.2: The score table of the different instances for the concept "apple"

<table>
<thead>
<tr>
<th>instances</th>
<th>score</th>
</tr>
</thead>
<tbody>
<tr>
<td>fruit</td>
<td>0.58</td>
</tr>
<tr>
<td>fresh fruit</td>
<td>0.076</td>
</tr>
<tr>
<td>silicon valley titan</td>
<td>0.061</td>
</tr>
<tr>
<td>fruit tree</td>
<td>0.058</td>
</tr>
<tr>
<td>hard fruit</td>
<td>0.055</td>
</tr>
<tr>
<td>company</td>
<td>0.043</td>
</tr>
<tr>
<td>tree fruit</td>
<td>0.037</td>
</tr>
<tr>
<td>climateric fruit</td>
<td>0.035</td>
</tr>
<tr>
<td>design oriented firm</td>
<td>0.03</td>
</tr>
<tr>
<td>tech company</td>
<td>0.024</td>
</tr>
</tbody>
</table>

The large majority of the concepts in the Microsoft Concept Graph concerns elementary concepts such as countries or cities. Those concepts are present in many taxonomies nowadays. However, Microsoft Concept Graph contains concepts way more accurate. As we have seen in the previous example, “apple” is a “design oriented firm” which is already very specific.

The data was extracted according to two methods: the syntactic or semantic iteration.

The syntactic method can be iterative. It begins with a set of examples or/and patterns. Then the set of patterns is expanding with the examples. With new patterns, the set of examples is also increasing and so on. For instance, we have the Hearst patterns for the relation “IsA” as described in [9] in the table 4.3. NP refers here to Noun Phrase.

<table>
<thead>
<tr>
<th>Hearst patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>NP such as (NP,)* (or</td>
</tr>
<tr>
<td>NP (, NP)* (,</td>
</tr>
<tr>
<td>NP(</td>
</tr>
<tr>
<td>NP(</td>
</tr>
</tbody>
</table>

Table 4.3: The different Hearst patterns

With those patterns, we can retrieve information from raw text. For instance, “Scandinavian countries, including Sweden...” indicates that Sweden is a Scandinavian country.

Focusing only on a syntactic approach has its downsides. First of all, natural language is ambiguous. We can easily create a sentence which will generate a false assumption with the Hearst patterns. For instance, the sentence “… countries other than Scandinavian countries such as France …” will lead to the relation “France is a Scandinavian country”.

One of the other problems is called semantic drift. Syntactic patterns obtained from examples may be of poor quality. For instance, we can find that the new pattern “in competition with x” from a set of football teams. However,
The recall is very low for the benefit of the precision. This is due to the fact that syntactic patterns are bad. One way of doing it is for example to take only a noun after a syntactic pattern and not a noun phrase. In the sentence “Scandinavian countries such as Sweden and Norway”, only ”Sweden” will be extracted.

Semantic iteration enables to take into account the knowledge already acquired by the taxonomy. For instance with the sentence “Scandinavian countries other than Norway such as Sweden”, the algorithm knows that there are two options “Sweden is a Norway” or “Sweden is a Scandinavian country”. From the taxonomy, it knows that “Sweden is a country” and that “Scandinavian country is a country” and therefore the sentence “Sweden is a country” is more frequent than the sentence “Sweden is a Norway”. This can also be applied to noun phrases including the word “and” or “or”. A relation “instance is a concept” is called an edge.

When a large set of relations ”IsA” has been created by using the previous methods, we have to build the taxonomy. Every element of the taxonomy is called a node. A node is either a concept (country) or an instance (Sweden). The instance may also be a subconcept (Scandinavian country). The next step is to merge nodes with each other. Some words are polysemous and thus can be linked to several concepts. Those several concepts are not related even if they have some instances in common. For example, the instance “apple” is linked to the concept “fruit” and to the concept “company”. However, the two edges should not be connected to each other.

With some properties, it is possible to get and merge some of the nodes created with the syntactic and semantic iterations. If the instances are present in the same sentence, we assume that they have the same meaning. The second merging technique is called the horizontal merge. If two sentences have instances in common, we assume that all the instances present in both sentences have the same meaning. The last method is called the vertical merge. This is can be illustrated by the following sentences : “countries such as Scandinavian countries, Sweden and Norway” and “Scandinavian countries such as Sweden and Norway”. This means that “Scandinavian country” is a sub concept of “country” and thus “Sweden” and “Norway” are instances from both concepts.

With the points previously discussed, it is possible to build a large taxonomy directly from raw texts. This is how the taxonomy Probase has been built. Probase is the previous version of Microsoft Concept Graph.

ConceptNet

One of the problems of Microsoft Concept Graph and of many other knowledge graph is that they are only available in English since it is trained on English
texts. The methods previously explained are appropriate for generating a knowledge graph in other languages, but it requires huge amount of texts that can be difficult to gather.

One of the other weaknesses of the techniques used is that they rely only upon unsupervised data. For a model, learning from unsupervised data has the advantage that the model is good for learning from any kind of data, even though it does not always work great.

At first, the idea behind ConceptNet [10] can be explained by the project called Open Mind Common Sense. This project was aiming at gathering annotated data from humans. Many other sources of data has been added to the process. The other sources of information are Wiktionary in several languages, Open Multilingual WordNet, JMDict which is a japanese dictionary. There was also a subgroup of DBPedia.

There are many relations in ConceptNet: Antonym, Distinct From, Etymologically Related To, Located Near, Related To, Similar To, Synonym, At Location, Capable Of, Causes, Causes Desire, Created By, Defined As, Derived From, Desires, Entails, Form Of, Has A, Has Context, Has First Subevent, Has Last Subevent, Has Prerequisite, Has Property, Instance Of, Is A, Made Of, Manner Of, Motivated By Goal, Obstructed By, Part Of, Receives Action, Sense Of, Symbol Of and Used For. We notice that some of the relations are symmetrical while others are not. Some relations are more developed than others since the sources of information are not evenly distributed and not contribute at the same relations.

One of the specificities of this knowledge graph is that the relations can happen between words of different languages. For example, we can have the triplet \((\text{Sweden}, \text{Synonym}, \text{Svenska})\). This triplet is called a node in the terminology of ConceptNet while the extremities of a node are called edges. Thus, those edges can be very helpful in case we want to use the base in different languages.

The size of ConceptNet is huge and it is available in many languages as you can see in the table 4.4. The presented languages are considered as the core languages, meaning that ConceptNet is describing it correctly. In all, there are 68 languages with more than 10000 edges in ConceptNet, such as Latin or Old English. The number of languages is even increasing until 304 if we are counting every language with more than 300 words. There are around 21 millions edges for 8 millions of nodes.
<table>
<thead>
<tr>
<th>Language</th>
<th>Size of the vocabulary</th>
</tr>
</thead>
<tbody>
<tr>
<td>English</td>
<td>1803813</td>
</tr>
<tr>
<td>French</td>
<td>3023144</td>
</tr>
<tr>
<td>Italian</td>
<td>1078629</td>
</tr>
<tr>
<td>German</td>
<td>825741</td>
</tr>
<tr>
<td>Spanish</td>
<td>782760</td>
</tr>
<tr>
<td>Russian</td>
<td>680205</td>
</tr>
<tr>
<td>Portugues</td>
<td>473709</td>
</tr>
<tr>
<td>Japanese</td>
<td>363663</td>
</tr>
<tr>
<td>Dutch</td>
<td>267641</td>
</tr>
<tr>
<td>Chinese</td>
<td>242746</td>
</tr>
</tbody>
</table>

Table 4.4: Sizes of the different core languages of ConceptNet

You can find a graph 4.3 with typical relations generated with ConceptNet.

Figure 4.3: Illustration of the node "ConceptNet" taken from the website conceptnet.io
Word embedding from knowledge graph

The method of building a word embedding is described in the paper [11]. Every element of the matrix is depicting the link between two edges present in the knowledge base. The cells are computed as the sum of all weights linking two edges. Then, this matrix can be reduced to a dimension of 300 in order to be manipulated more easily.

The weights are not provided in Microsoft Concept Graph. However, they exist in ConceptNet for every node. The matrix of word embeddings obtained from ConceptNet is called ConceptNet-PPMI.

This embedding is however not really efficient. There is no training phrase with this method. The embedding did not learn anything from a text. Thus, for a given word, ConceptNet-PPMI will not save any information of context. Moreover, for saving time of computations, the algorithm discarded all the edges connected to three edges or less.

To respond to this problem, a method exists in which an existing word embedding is enhanced with a knowledge graph. This method is called retrofitting and has been implemented with the ConceptNet knowledge graph. It consists in computing and refining a word embedding matrix using a knowledge graph. Thus, every new word vector will be close to their former word vector, but there will be also close to their neighbours in the knowledge graph. The new word vectors are obtained by minimizing the following function:

$$\Psi(Q) = \sum_{i=2}^{N} \left[ \alpha_i \| q_i - \hat{q}_i \|^2 + \sum_{(i,j) \in E} \beta_{ij} \| q_i - q_j \|^2 \right]$$  \hspace{1cm} (4.9)

$\hat{q}_i$ are the word vectors that we are looking for through the retrofitting, $q_i$ are the word vectors of the original embedding, $E$ is the set of the all possible edges between the word of index $i$ and word of index $j$. In the case the word at index $i$ does not exist in the original embedding, $\alpha_i$ is equal to 0. The $\beta_{ij}$ are referring to the weights present between two edges in ConceptNet.

One of the advantages of this method upon other word embeddings is that it is learning from a knowledge base including many languages. Usually, a word embedding is only based on text only in one language. Here, the model will give an embedding of diverse languages, even though the first embedding is only base on one unique language at first. The model will learn from the synonyms in other language for each word, giving additional information. Retrofitting has also the advantage of being quite easy to implement. Indeed, only an existing embedding is required and not the whole data that was necessary to generate the embedding.

ConceptNet provides several pre-computed word vectors. Those word embeddings are called ConceptNet Numberbatch are available in English or in multilingual [12]. According to the paper [11], these word embeddings outperform the other popular word embeddings GloVe, FastText and Word2Vec at some evaluating tasks. The researchers also claim that word vectors from ConceptNet Numberbatch were less stereotyped than the word vectors from other
embeddings.

4.1.5 Relationship between the models

Every unsupervised based method will have some similarities since all these methods are counting the occurrences for each word. Unsupervised methods are very convenient since there are large corpus of words or documents on the Internet while large amounts of supervised data in natural language are quite hard to gather. It is also difficult to define the best supervised task for sentence representation learning and even with a good supervised task, it will be hard to find a dataset large enough for the neural network required. Another problem of supervised task is that a model generated from a supervised task will perform well on what it was trained but poorly on other task. A model in NLP will therefore forget partially the semantic meaning.

4.2 Sentence embeddings

As an extension of the definition of word embedding, we define a sentence embedding when the vocabulary contains sentence, or at least groups of word instead of single word. There are several methods to get a sentence embedding.

4.2.1 Mean embeddings

The simplest method for generating a sentence embedding is to take the mean of the sum of every word embedding of the sentence. This method does not require any additional training and thus is very interesting for its low computational cost and its speed.

4.2.2 Word embeddings with TF-IDF

One other method is to use the TF-IDF, standing for Term Frequency-Inverse Document Frequency [13]. The TF-IDF is a numerical statistic. Its goal is to give for any word a score to indicate the degree of importance of the word in the sentence. Thus, this score can be used as a weight in our sentence vector computations. The scores computed depend on the document on which the TFIDF has been computed.

The term frequency $tf(w, d)$ of the word $w$ in the document $d$ can be defined of several manners. The easiest method consists to use the number of times that the word $w$ appears in the document $d$. Mathematically, the tfidf is described as follows.

$$
tfidf(w, d) = tf(w, d) \log \left( \frac{N}{df_w} \right) \quad (4.10)
$$

where $w$ is a word, $d$ is a document, $N$ is the total number of documents and $df_w$ is the number of documents that contains $w$. 

19
This score takes into account the frequency of the words inside a document. Vector from very common words will be granted a smaller weight while rarer words will receive a higher weight. One advantage is that it adapts itself in function of the documents that it is receiving.
Chapter 5

Clustering

One well-known application of word and sentence embeddings is text classification.

Text classification can be supervised or unsupervised. Supervised text classification is known as text categorisation. Text categorisation consists in classifying documents into specific categories. These categories are defined beforehand and their number $K$ is set. Therefore, we have a training set. Conceptually, we want to know the best function $\phi$ minimizing the function $R$:

$$ R(\phi) = E[Y \neq \phi(X)] \simeq f(X) $$

(5.1)

$X_i$ are denoting the observations and $Y_i$ are denoting the labels for these observations, with $K$ different labels. When the function $\phi$ is found, we will be able to predict clusters for new different labels.

Unsupervised text classification is known as text clustering. The aim of text clustering is to sort the documents into different classes as well, except that we do not know what the different categories and how many they are. Moreover, contrary to text categorisation which is a supervised problem, text clustering is unsupervised. Therefore, this task is more difficult than text categorisation.

Conceptually, we only have the set of points that we want to classify: $D_n = (X_1, \ldots, X_n)$. The number of clusters $K$ is unknown and we aim for a partition of the dataset $D_n$:

$$ \Delta = (C_1, \ldots, C_K) $$

(5.2)

There are two types of clustering results: hard clustering and soft clustering.

Hard clustering consists in stating that an observation belongs to one and only one cluster. Therefore, we are able to get the partition $\Delta$.

Soft clustering will not state that an observation belongs to one specific cluster, but will assign to each point the distribution of the probabilities to be part of a cluster $C_i$:

$$ P[X_i \in C_k] = \gamma_{k,i} $$

(5.3)

with $\sum_{k=1}^{K} \gamma_{k,i} = 1$
Depending on the observations, it is sometimes difficult to create a decent clustering, since two clusters can be very close to each other, or even be overlapped. In this case, the soft algorithm can be a good solution because it enables to tell what are the most probable clusters for an observation instead of telling only the most probable.

We can classify clustering algorithm into two categories: the compactness-based algorithms and the connectivity-based algorithms.

An algorithm based on compactness structure of the data will assign a cluster to a certain point according to its distance to the cluster centroid. This distance can be obtained by several methods. Therefore, two close points will end up in the same cluster because they will have the same closest centroid cluster. One example of compactness-based clustering algorithm is the K-means algorithm.

From the other side, a clustering based on connectivity will connect the points that are close from each other. Those points will be classified in the same cluster. In the end, two points from a same cluster may be further away than one of these points and one point from another cluster. One example of connectivity-based algorithm is the spectral clustering algorithm.

In the first section of this Chapter, different types of clustering methods are presented while some measures for evaluating a clustering are proposed in the second section.

5.1 Clustering algorithms

5.1.1 K-means Algorithm

The K-means algorithm has been introduced for the first time by H. Steinhaus in 1957 [14]. K-means is an iterative clustering algorithm. It only requires one parameter which is the number $k$ of clusters.

Algorithm

At first, we take $k$ points in the vector space. Those points are randomly chosen and will be the centroids of the future clusters. Afterwards, all vectors are assigned to the closest centroid. Thus, each centroid forms a cluster. But as the initial centroids were randomly chosen, the clusters generated at first are really bad. Therefore, we get a new centroid for each cluster by computing the mean on every vector related to the specific cluster. The distance between the old and the new cluster centers is also computed. And we can go to the next iteration.

The algorithm is ending when the new clusters are almost at the same place as the previous iteration. The K-means algorithm gives evenly distributed clusters.
Advantages

K-means algorithm is fast to implement. The complexity of the algorithm is estimated to $O(iknd) = O(n)$ where $i$ is the number of iterations, $k$ the number of clusters, $n$ the number of vectors and $d$ the dimensions of the vectors.

Drawbacks

The number of parameters for K-means algorithm is a downside of the algorithm. Indeed, it can be tricky to find the right $k$. Some methods will be explained below. One of the other problems is that the clusters found with K-means algorithm are always spherical. The figures 5.1 show this limit of the K-means algorithm.

![Figure 5.1: One limit of the K-means algorithm](image)

(a) Set of points depicting two circles  (b) Clusters found by the K-means

As human, we can clearly see that there are two clusters. Each cluster corresponds to a circle. However, K-means algorithm will split the two circles into two equal parts. For instance, there will be one part of each circle in the first cluster, and the other part of each cluster in the second cluster. This means that K-means algorithm considers that all clusters have the same variance to their centroid, which can be wrong in practice.

The third drawback is that the K-means algorithm is not deterministic. Indeed, the initialization is random and can lead to different clusters in the end.

Estimation of the number of clusters

As we have highlighted before, K-means algorithm does not find the optimal number of clusters. This number has to be estimated. A too large number of clusters will cause an overfitting the data while too few clusters will not give us any additional information about the data. The usual method is to run the algorithm over a range of different values of $k$ and then to compare the results. One way for comparing the results for different $k$ is to look at the sum of squared distances of points to their cluster center. The goal is then to minimize this value. However, we cannot look at this metric as the only way to determine
which $k$ we have to choose because as the number of clusters increases, the points will be closer to their cluster center. Indeed, this sum is minimized for a number of clusters $k$ equal to the number of vectors in the dataset. The solution is to plot the sum of squared distances of points to their centroid as a function of $k$. Then, we chose the point where the slope of the curve begins to decrease with less intensity. This point is called the elbow point.

This point can be found by computing the second derivative of the function. One of the other methods is to compute for each $k$ the angle value formed by the curve. We keep the $k$ when the angle value is below a certain predefined threshold. However, this method has not provided the best result for word vectors.

The drawback of this method is that it needs to run several times the K-means algorithm. Thus, the time gained by using this fast algorithm is lost by running it several times.

**Improvement**

The first point of improvement concerns an improvement in term of clustering results. K-means requires a random initialization at its very beginning. Therefore, the K-means algorithm may lead to different results when we run it several times. Taking only one result of clustering will thus be quite risky, since it might be very different from the usual result. Thus, repeating several times K-means and taking the means of the groups obtained will be more efficient and safer. This will however increase the computation time.

The second point of improvement is detailed in the following part.

### 5.1.2 Minibatch K-means

Another version of K-means algorithm has been proposed [15] by D. Sculley and is referred to as minibatch K-means algorithm.

**Algorithm**

The idea behind minibatch K-means is that instead of taking the whole dataset for each iteration of the algorithm, it takes only a small portion of the whole data. The size of these batches is predefined in the parameter of the minibatch K-means algorithm. The data points of the batches are chosen randomly among all points of the dataset. While the usual K-means is updating the centroids at each iteration over the whole dataset, minibatch K-means updates the centroids after each batch.

**Advantages**

Minibatch K-means is faster than the normal K-means algorithm, especially for very large dataset.
Drawbacks

The results of the clustering are slightly less good than with the usual K-means algorithm. However, the number of points impacted by the loss of quality is low.

5.1.3 Graph linkage clustering

One other method to make a cluster analysis is to make a hierarchical clustering and has been introduced by Joe H. Ward in 1963 [16] in its one of its possible forms. This method is also called the graph linkage clustering. This method enables to create a hierarchy between the clusters. There are two types for generating hierarchical clustering: the agglomerative type and the divisive type. The agglomerative hierarchical clustering consists in generating a set of different clusters upon all the dataset, and then in merging clusters in pairs until there is only one cluster left. The initial set of clusters can be the set of all the observations. The divisive hierarchical clustering is reversed. There is only one cluster at the beginning. This cluster is then split in several clusters afterwards.

In the case of agglomerative clustering, two clusters are merged if their dissimilarity is the lowest. This dissimilarity can be computed of different manners according to the linkage criteria chosen.

In the case of divisive clustering, a cluster is split according to a classic clustering algorithm like K-means. In order to choose which cluster needs to be split, the average dissimilarity of each cluster is computed. The cluster with the highest dissimilarity is chosen. Thus, the divisive clustering is requiring more computations since it needs to run the K-means algorithm at each iteration.

Some of the existing linkage criteria are the following. The single linkage aims to minimize the distance between the observations of each cluster and its neighbours, i.e. \( \{d(x, y) : x \in X, y \in Y\} \). In the case of agglomerative clustering, it means that the two clusters minimizing the criteria are chosen and then are merged. The complete linkage is intended to minimize the maximum distance between the observations of each cluster and its neighbours. The average linkage looks at the average of the distance between the observations of two clusters while the ward linkage is minimizing the total variance of each cluster.

The single linkage can often lead to wrong merge of clusters. Indeed, only looking at the closest observations between two clusters can be wrong since two clusters can be well separated but present only two points that are very close from each other. In this case, the two clusters are merged while there was no reason for it.

Advantages

The results of a graph linkage clustering give more information than usual for a clustering algorithm. Indeed, with the dendrogram, we know what are the most similar clusters and the most dissimilar ones.
Drawbacks

The hierarchical algorithm has a higher complexity than the K-means algorithm. The time complexity is between $O(n^2 \log(n))$ and $O(n^3)$ according to the chosen linkage chosen. This is due to the fact that the algorithm looks at each iteration for the best candidate to split or for the best candidates to merge.

5.1.4 Gaussian Mixture Models

Gaussian Mixture Models can also be used to perform cluster analysis. It consists in finding a combination of several Gaussian distributions that fits the best the data. Conceptually, the data distribution can be written as follows:

$$p(x) = \sum_{k=1}^{K} \pi_k N(x|\mu_k, \Sigma_k)$$

with $\sum_{k=1}^{K} \pi_k = 1$

Contrary to the other methods tackled previously, the Gaussian Mixture Model is a soft clustering algorithm. Gaussian Mixture Models can be seen as an improvement of the K-means clustering, since the centroids of the K-means algorithm can be considered as the means of the different Gaussian distributions. The covariance of the different Gaussian distributions is therefore adding some new information to the data.

Algorithm

The Gaussian Mixture Models are learned from the Expectation Maximization Algorithm, which was introduced in this paper [17] in 1977.

The process is quite similar to the K-means algorithm. First, it randomly assigns every sample of the dataset a cluster. It can thus compute the estimation of the mean $\widehat{\mu}_k$ and the variance $\widehat{\Sigma}_k$ of each cluster $C_k$. The estimated weights $\widehat{\pi}_k$ are set identical and equal to $\frac{1}{K}$.

The next step is the expectation step. The expectation of the temporary cluster $C_k$ for each sample $X_i$ is computed according to the means $\widehat{\mu}_k$ the variances $\widehat{\Sigma}_k$ and the weights $\widehat{\pi}_k$. Mathematically, it is finding the following expression:

$$\widehat{\gamma}_{ik} = \frac{\widehat{\pi}_k N(x_i|\widehat{\mu}_k, \widehat{\Sigma}_k)}{\sum_{l=1}^{K} \widehat{\pi}_l N(x_i|\widehat{\mu}_l, \widehat{\Sigma}_l)}$$

Afterwards, there is the maximization step. The expectation previously computed is maximized in order to update the means $\widehat{\mu}_k$ the variances $\widehat{\Sigma}_k$ a,d the weights $\widehat{\pi}_k$. 

26
\[
\hat{\pi}_k = \frac{\sum_{i=1}^{N} \gamma_{ik}}{N} \quad (5.6)
\]

\[
\hat{\mu}_k = \frac{\sum_{i=1}^{N} \gamma_{ik} x_i}{\sum_{i=1}^{N} \gamma_{ik}} \quad (5.7)
\]

\[
\widehat{\Sigma}_k = \frac{\sum_{i=1}^{N} \gamma_{ik}(x_i - \hat{\mu}_k)^2}{\sum_{i=1}^{N} \gamma_{ik}} \quad (5.8)
\]

The expectation and maximization steps are repeated until the algorithm converges. This convergence is defined from a threshold \(\epsilon\) on the parameters \(\hat{\mu}_k\), \(\widehat{\Sigma}_k\) and \(\hat{\pi}_k\). The probabilities for each point \(x_i\) to belong to the cluster \(C_k\) are equal to \(\gamma_{ik}\) at the end of the process.

**Advantages**

One of the interests of using a Gaussian Mixture Models is that it can learn more about the data than the K-means algorithm, notably in term of the covariance and the weight that each cluster can have. One of the other advantages of the algorithm is that it provides information on overlapped clusters. Indeed, as the algorithm gives a soft clustering, it will be easier to detect which clusters are more closely related than others.

**Drawbacks**

As the K-means algorithm, the number of clusters is a parameter and not a result of the algorithm.

Even though the algorithm is for quite general use, it assumes that the data can be described as a multivariate Gaussian which is not always true.

### 5.1.5 DBSCAN

DBSCAN stands for Density-Based Spatial Clustering of Applications with Noise and has been introduced in 1996 by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu [18]. The idea behind this algorithm is to use a density parameter in order to group the close points together. To be highly efficient, the dataset has to display regions of high density of points simultaneously with regions of low density.

The DBSCAN algorithm has two parameters. The first parameter is referred to as \(\epsilon\). It depicts a distance. This distance is the longest distance for which two points are considered to be neighbours. The second parameter is \textit{minPoints}. This parameter corresponds to the minimal number of neighbouring points to form a cluster. Contrary to some other clustering techniques, such as K-means algorithm, the DBSCAN does not require the number of clusters \(k\) to find. This number will be automatically generated by the algorithm.
Algorithm

The DBSCAN algorithm begins with a point arbitrary chosen. This point should have at least \( \text{minPoints} \) neighbours whose distance from the starting point is lower than the parameter distance \( \epsilon \). From this point, the associated cluster will be generated. More specifically, the algorithm will find the points whose distance from the starting point is lower than the parameter \( \epsilon \). Those points form the first cluster. For each new point, the algorithm inspects their neighbours within \( \epsilon \) distance. With less than \( \text{minPoints} \) neighbours, the process is ended for this point. The point is classified in the border points, i.e. the boundary points of a cluster. However, with more than \( \text{minPoints} \) neighbours, the process is resumed with the neighbours added to the growing clusters. The point is then classified into the core points of a cluster. When for every point of the formed cluster does not have enough neighbours with a distance smallest than \( \epsilon \), the process is completed and the first cluster is created. Every point of the cluster will not be used in the following steps of the algorithm.

A next cluster is created with the same process by starting with a new point which is not included in the first cluster. The algorithm ends when every point has been allocated a cluster. However, some points have not received any cluster. Indeed, there may be points having less than \( \text{minPoints} \) neighbours with a distance smaller than \( \epsilon \). Those points are then considered as noise by the algorithm. They do not belong to any cluster. In the end, every point belongs either to a cluster or to the noise points.

Advantages

As highlighted before, the number of clusters \( k \) is not a parameter for the DBSCAN algorithm. This parameter can be tricky to find for the algorithm which are requiring it. One of the upsides of the algorithm compared to the K-means algorithm is that DBSCAN is not iterative. When a point has been allocated a cluster by the DBSCAN algorithm, this cluster never changes. It is also addressing the problem of the outliers by dismissing every noisy point, which is very interesting.

Drawbacks

Like the K-means algorithm, the DBSCAN algorithm is not fully deterministic. Indeed, according to the order in which the points are processed, the clustering results may be a bit different. The points at the limit between two clusters can be classified in either of the two. This problem is however not raising in many datasets.

One assumption made by the DBSCAN algorithm is that all clusters have the same density. There is no reason why that can be true every time and may lead to wrong results at some time.

One of the other problems of DBSCAN algorithm occurs if some clusters in the dataset overlap each other. In this case, due to the fact that some elements
of both clusters are close to each other, the DBSCAN algorithm will classify the
two clusters into only one.

Even though the number of clusters $k$ is not a parameter of the DBSCAN
algorithm, there are two parameters needed. Those parameters can be hard to
choose in practice. For instance, if the parameter $\epsilon$ is taken too large, there may
be only one big cluster covering all the dataset. From the other side, with a
too small $\epsilon$, many points will be considered as noise and will not belong to any
cluster, leading to a poor clustering. There is also a third parameter needed :
the measure of the distance chosen. This choice is especially important when
the data has a high dimension.

5.2 Relevance of a clustering on text data

Judging the relevance of a clustering result can be tricky, especially for vectors
depicting words or sentences. We will see in this part what are the properties
defining a good cluster.

The best way to see if a clustering method has been correct or not is to per-
form a clustering on a labelled data. Then, a comparison can be done between
the true labels and the predicted labels. The recall and the precision between
the predicted clusters and the real clusters can be found for instance.

The problem for the clustering of the text data is that there is no labelled
data available for that. Furthermore, it is really hard to tell what can be a good
clustering on some datasets. There might even be several decent clusterings
possible for a same dataset.

5.2.1 The silhouette coefficient

The silhouette score was introduced in 1987 by Peter Rousseeuw [19]. It uses
two scores and is computed for each point of the dataset. The first score $a$
corresponds to the mean of the distance point $X_i$ and every element of its own
cluster $C_k$. This score $a$ is called the average dissimilarity of the point $X_i$ with
every points of its cluster. The second score $b$ corresponds to the mean of the
distance between a point $X_i$ and every element of the closest cluster $C_k'$. From
these two scores, the silhouette score for a given point $X_i$ is defined as:

$$s(X_i) = \frac{b - a}{\max(a, b)}$$  \hspace{1cm} (5.9)

A good clustering will give a high silhouette score while the score will be low
in the case of a poor clustering. We can notice that the score is limited between
-1 and 1. Let’s discuss the different possible scenarios.

If the score is equal to 0, it indicates that the two clusters are overlapping.
Indeed, this case implies that the metrics $a$ and $b$ are equal and thus that the
elements of the cluster $B$ are as close as the elements of the cluster $A$ for the
element $i$. 

29
A score equal to -1 points that the clustering is bad. In this instance, the element \(i\) of cluster \(A\) is closer to the elements of cluster \(B\) than to the elements of cluster \(A\) because \(b\) is equal to 0.

If the silhouette score is 1, the clustering is good. It means that the element \(i\) of cluster \(A\) is much closer to the elements of its own cluster \(A\) than to the elements of the cluster \(B\), leading to a score \(b\) largely greater than a score \(a\).

The silhouette coefficients for all the points of the dataset will give a representation on how well the different clusters are separated. It is also possible to find the best number of clusters \(k\) thanks to the silhouette graph. For doing that, the silhouette score is computed for every point of each cluster. Then, the silhouette score average is computed. If in one or more clusters, every element is below the silhouette score average, we can conclude that the clustering hasn’t been successful and that the number of clusters is not the best.

The mean of the silhouette score for all points of the dataset can also give an idea on how well the clustering performed. It is possible to determine the number of clusters \(k\) needed for a clustering with the average silhouette score. It can be done by computing the average score on a scale of different values of \(k\). The value of \(k\) kept is the value for which the score has been the higher.

### 5.2.2 Calinski-Harabaz Index / the variance Ratio Criterion

Like the silhouette coefficient, the Calinski-Harabaz Index is a good method to describe how well your clustering worked if you don’t have any true labels to compare with. It has been introduced by Caliński, T. and Harabasz, J. [20]. It depends on the number \(k\) of clusters.

It is defined as follows:

\[
s(k) = \frac{\text{Tr}(B_k)}{\text{Tr}(W_k)} \times \frac{N - k}{k - 1} \quad (5.10)
\]

with the between group dispersion matrix

\[
B_k = \sum_{i=1}^{k} n_i (c_i - c)(c_i - c)^T \quad (5.11)
\]

and the within-cluster dispersion matrix

\[
W_k = \sum_{i=1}^{k} \sum_{x \in C_i} (x - c_i)(x - c_i)^T \quad (5.12)
\]

\(N\) is the number of points in the datasets, \(C_i\) the points of the cluster \(i\), \(c_i\) the centroid of the cluster \(i\), \(n_i\) the number of points in the cluster \(i\) and \(c\) the center of the whole dataset.

The higher the Variance Ratio Criterion, the better the clustering. Indeed, the clustering is better if the different clusters are well separated and so well dispersed. Moreover, a cluster is well defined if it is dense.
5.2.3 Davies-Bouldin Index

This is another method to estimate the validity of the results of a clustering [21]. This coefficient counts on the calculation of the similarity between the clusters. For each cluster, the similarity is computed with its most similar cluster. The similarity is expressed as follows:

\[ R_{ij} = \frac{D_i + D_j}{d_{ij}} \]  

(5.13)

where \( D_i \) is the mean distance between the points of a cluster and its center and \( d_{ij} \) is the distance between the centers of clusters \( i \) and \( j \).

Finally, the Davies-Bouldin Index is described by the following expression:

\[ DB = \frac{1}{k} \sum_{i=1}^{k} \max_{i \neq j} R_{ij} \]  

(5.14)

The goal is to have the lower score as possible. A very low score indicates that every cluster is positioned far away from the other clusters, or more precisely that every cluster is quite far from its closest and influential enough neighbour. This influence is described here by the diameter of the clusters.
Chapter 6

Implementation

In this Chapter, we will implement the different solutions over different word embeddings and different type of clusterings.

6.1 Working framework

6.1.1 Input data

The data used during the thesis at Askhub was confidential. Therefore, the computations cannot be done on real chatbots data and we have to choose other datasets. The dataset chosen must be the most similar to the data generated by a chatbot. The typical requests of message users to a chatbot are very short messages.

Also, a chatbot dataset is often very specialized in term of contents. A chatbot rarely has a very wide range of possible answers. Usually, "greetings" is the most commonplace messages. Then, the next most usual messages are about a service that the bot is providing, like booking a hotel room for a chatbot specialized for a chatbot from an online booking website.

The data chosen is taken from a large dataset of 1600000 tweets. This dataset was generated by a group of researchers in 2012 [22] who realized a sentiment analysis over it through supervised methods. The tweets are taken from April 6, 2009 to June 25, 2009. Tweets are a good compromise for chatbot data. Indeed, a tweet must be shorter than 140 characters which is comparable to the length of a request of a chatbot. Furthermore, even though the subject of a tweet can be very diverse, there are always converging topics of discussion.

For practical reasons, the actual computations have been done over a subset of 200000 tweets of the large dataset.
6.1.2 Cleaning

The data was containing many web links. Therefore, those links have been removed in order to not interfere with the clustering results. Indeed, there was a significant risk that all requests with a link will be found in the same cluster. A spellchecking has also been done in order to improve the vectorization step.

6.1.3 Choice of the word embeddings

For creating the word embeddings, we have chosen to use pretrained English word vectors from GloVe [5] and ConceptNet Numberbatch [12]. This is done in order to be able to compare the different results in function of the embedding taken. The dimension of the GloVe vectors is 200 while the dimension of the ConceptNet Numberbatch vectors is 300.

The pretrained model of ConceptNet Numberbatch has been created applying the retrofitting method discussed in the section 4.1.4. For training their model, they used the word vectors from the model Word2Vec and GloVe. The necessary weights were generated by their knowledge base ConceptNet 5.5.

Intuitively, we would say that the pretrained vectors from ConceptNet Numberbatch will have better performance than the GloVe embeddings, since that embedding is based on more data and is taking advantage of the relations between words from its knowledge base. Furthermore, the ConceptNet Numberbatch vectors have more features inducing to a better representation of the data.

6.1.4 Clustering methods

To determine what it is the most suitable method of cluster analysis for word classification, the clustering methods presented in Chapter 5 have been performed in the following section. Each cluster has been named by taking its three most frequent words.

6.2 Results

6.2.1 Identification of the number of clusters through K-means clustering

To determine the number of clusters in the dataset, we first ran the K-means minibatch algorithm with values of k ranging between 1 and 70. In the figure 6.1 we can see the distortion in function of the number of clusters.
The elbow point is taken for $k = 9$. For the clustering algorithms requiring as parameter the number of clusters $k$, this number will be chosen.

6.2.2 K-means

As the cluster analysis of K-means and K-means minibatch are roughly the same, the computations have been done by using this last one. The figures 6.2 and 6.3 are depicting the silhouette score through the different clusters with the two different embeddings.

As we can see, the silhouette score seems pretty bad in average. Many clusters have a silhouette score below 0, which means that they are overlapped with an other cluster.

We can notice some similarities between the two cluster analyses even though the clusters are not in the same order. The clusters 3 with GloVe embeddings and 2 with ConceptNet Numberbatch are constituted mostly by the same tweets. The clusters 4 and 5 of GloVe embeddings and the cluster 5 of ConceptNet Numberbatch embedding are denoting the tweets written in a language different from English.
We also remark that some clusters seem to deal with the same subject. For instance, the clusters 0 and 2 generated with ConceptNet Numberbatch are constituted of tweets dealing with job offers. On the other hand, some clusters seems to be too large and would need to be split like the cluster 6 whose tweets are very diverse.
6.2.3 Gaussian Mixture Models

The silhouette score has been computed for each tweet in the figures 6.4 and 6.5.

The silhouette score looks bad, with very few clusters whose the average silhouette score is above 0. This can be partially explained by the fact that different clusters are covering job related tweets, leading to overlapping clusters. Once again, there are some similarities between the two clustering analyses.
The predicted posterior probability $P(C_k|x)$ illustrating that a vector $x$ is in the cluster $C_k$ have been computed. Unfortunately, those results do not bring any valuable information about the clustering, since the probabilities observed are $P(C_k|x) = 1$ for the chosen cluster and $P(C_k|x) = 0$ for the others.

### 6.2.4 DBSCAN

The DBSCAN algorithm gives a large number of clusters for both GloVe and ConceptNet Numberbatch embeddings, 140 and 214 respectively. Furthermore, the clusters found are not satisfactory. One cluster predominates all the other, with more than 90 percent of the tweets falling in this cluster. Almost all clusters are otherwise only constituted of one sentence, sometimes repeated multiple times. This is probably due to the fact that DBSCAN algorithm is designed to find clusters of same density level.

One solution may be to take a higher $\epsilon$ and a lower $\text{minPoints}$ in order to merge the closest clusters. The problem is that the dominating cluster will be then even bigger.

### 6.3 Comparisons between the results

In this part, we will compare the different cluster results. The table 6.1 summarizes the different clustering done with several metrics to evaluate the clustering performances.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Embedding</th>
<th>Silhouette</th>
<th>Calinski-Harabaz</th>
<th>Davies-Bouldin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minibatch</td>
<td>GloVe</td>
<td>0.04</td>
<td>6342.46</td>
<td>3.61</td>
</tr>
<tr>
<td>Minibatch</td>
<td>ConceptNet</td>
<td>0.01</td>
<td>3522.62</td>
<td>4.31</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>GloVe</td>
<td>-0.40</td>
<td>0</td>
<td>2.60</td>
</tr>
<tr>
<td>DBSCAN</td>
<td>ConceptNet</td>
<td>-0.39</td>
<td>0</td>
<td>2.87</td>
</tr>
<tr>
<td>GMM</td>
<td>GloVe</td>
<td>-0.13</td>
<td>0.69</td>
<td>7.92</td>
</tr>
<tr>
<td>GMM</td>
<td>ConceptNet</td>
<td>-0.12</td>
<td>1.16</td>
<td>8.57</td>
</tr>
</tbody>
</table>

Table 6.1: Results of the algorithm with the different scores

As first remark, we can notice that the difference between the GloVe and ConceptNet Numberbatch embeddings are almost inconspicuous. The largest difference is for minibatch K-means algorithm with a large difference for the Calinski-Harabaz index.

The results of minibatch K-means seems better according to the metrics. It possesses a very high Calinski-Harabaz score, with a quite low Davies-Bouldin score. The silhouette score, on the other hand, is rather bad.

The DBSCAN results are bad overall. The low Calinski-Harabaz score can be explained by the observed imbalance between the size of the clusters, as explained in the section 6.2.4. The cluster gathering 90 percent of the tweets causes a very high within-cluster dispersion matrix $W$. The Davies-Bouldin
index is the best among the three types of clustering methods which can be explained since the number of clusters is high.

The Gaussian Mixture Model gives poorer results than the K-means algorithm for every metrics proposed.

One of the possible explanations for these results is that the metrics did not comprehend well the clusters created. As highlighted in the sections 6.2.2 and 6.2.3, the silhouette score was bad even though the cluster was not qualitatively bad. Indeed, the metrics proposed gives better results for convex clusters. Thus, if several clusters are covering the same subject, the different scores are poor. However, a bad score does not mean that the clustering is poor. The step of vectorization is also something important to keep in mind. If poor tweet vectors are generated, clustering these vectors will not work well.
Chapter 7

Conclusions and Future Work

7.1 Conclusions

As the different clustering methods were only performed on one dataset, the results of the study cannot be considered as significant enough to be generalized to more dataset. Nevertheless, it could be relevant to use them as a procedure to follow in order to perform clustering over short texts data.

7.1.1 Word embeddings

A reason explaining why the text clustering seems to be poor is that the initial word embedding was not good enough to perform a good text clustering afterwards. In this case, the best solutions may be to choose another word embeddings. The results showed that there was no significant difference between the two models tested GloVe and ConceptNet Numberbatch. Nonetheless, another word embeddings could potentially perform better.

7.1.2 Clustering algorithms

Different clustering methods have been proposed. The fastest method remains the K-means algorithm. The Gaussian Mixture Models can be interesting since it comprehends better the data. Unfortunately, these assumptions could not be constated through the results. The DBSCAN was not working as expected however.
7.2 Future Work

Even though the different scores were quite poor on the text data clustering, it does not mean that everything has failed. Indeed, classifying text data is a hard task, even for humans. For instance, how classify an ambiguous sentence, or more simply a sentence which deals with two different topics? These kinds of requests for chatbots are difficult to handle since they are usually designed to answer one question at a time.

In continuity of this master thesis, work on new metrics over cluster analysis could be done. One of the solutions envisaged to enhance the quality of the clustering could be to detect the poor clusters and then, to split or merge them in order to improve this metric.
Bibliography


