FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Text Classification using Unsupervised Learning techniques

Ricardo Henrique Teixeira Duarte

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Henrique Daniel de Avelar Lopes Cardoso

July 23, 2018

Text Classification using Unsupervised Learning techniques

Ricardo Henrique Teixeira Duarte

Mestrado Integrado em Engenharia Informática e Computação

July 23, 2018

Abstract

Due to the exponential increase of available data, information disorganization became a current problem. More important than gathering data, is to be able to organize it in order to facilitate an efficient information analysis.

This project's main goal is to present a solution for the problem of information disorganization by using unsupervised learning techniques to classify text documents into categories. More specifically, the main objective is to create a system that is able to organize text emails into folders and then extract topics related with each folder while applying different clustering methods in order to investigate their advantages and disadvantages in this classification context. Therefore, this project includes itself in a typical unsupervised text classification problem where the solution includes tasks such as text pre-processing (data cleaning), feature extraction and clustering followed by the evaluation metrics retrieval and the analysis of the obtained results.

The implementation of this solution uses technologies such as python programming language and *scikit-learn* and *gensim* libraries.

The dataset integrates real life data that includes six email inboxes with different sizes given by collaborators working at the company AMT-Consulting for research purposes.

The final result includes not only a prototype that, given a large amount of email data, is able to organize it into different labeled categories but also several conclusions about the different feature extraction methods and clustering algorithms regarding their advantages and disadvantages applied to the AMT-Consulting dataset.

This project has a certain degree of innovation because it solves the problem of unorganized text information in a very specific company context that includes certain characteristics that differ from other existing text datasets.

Resumo

Devido ao aumento exponencial da informação disponível, a desorganização desta tornou-se um problema atual. Mais importante do que obter informação, é organizá-la de modo a tornar mais eficiente a sua análise.

O objetivo principal deste projeto é apresentar uma solução para o problema da desorganização de texto usando técnicas de aprendizagem não supervisionada para classificar documentos de texto em categorias. Mais especificamente, o objetivo principal é criar um protótipo de um sistema capaz de organizar emails em pastas e, posteriormente, extrair tópicos relacionados com cada uma das pastas aplicando diferentes métodos de *clustering* para investigar suas vantagens e desvantagens nesse contexto de classificação.

Deste modo, este projeto inclui-se na categoria de problemas de classificação de texto não supervisionado, no qual a solução inclui tarefas como pré-processamento de texto, extração de *features* e uso de algoritmos de *clustering* seguidos da extração de métricas de análise de modelos criados.

Esta solução utiliza tecnologias como a linguagem de programação python e bibliotecas como *scikit-learn* e *gensim*. O *dataset* usado neste projeto integra dados reais distribuídos por seis caixas de entrada de correio eletrónico com diferentes tamanhos. Estes dados foram fornecidos por colaboradores da empresa AMT-Consulting para fins de investigação.

O resultado final deste projecto inclui não só um protótipo capaz de organizar emails em diferentes pastas, como também várias conclusões sobre os diferentes métodos de extração de *features* e algoritmos de *clustering* relacionadas com as suas vantagens e desvantagens aplicadas ao conjunto de dados da AMT-Consulting.

Este projeto tem um certo grau de inovação dado que este problema de categorização de texto inclui-se num âmbito muito específico que se inclui na categoria de organização de caixas de entrada a nível empresarial.

Acknowledgements

Firstly and most importantly, I would like to thank my parents, my sister and my brother-in-law for giving me all the conditions to finish not only this dissertation but also my whole education, for giving me an amazing support throughout this semester's worst times and for believing in me no matter what, sometimes even more than I believed in myself, and for that, I will be forevermore grateful.

Secondly, I must express how important my friends were in this path. The support during this work was immense and for all the moments that they kept me company during the work time and during the breaks, for all the good moments that we spent in our academic journey and the hard work moments, thank you.

Thirdly, I would to thank my supervisor Professor Henrique Lopes Cardoso for all the time he spent helping me with this work and for his patience when hearing my doubts.

Lastly, but not least, I would like to thank AMT-Consulting for creating the theme for this work and for providing the email datasets that were very important for this work's development.

Ricardo H. Duarte

‘Genius is 1% talent and 99% percent hard work...’

Albert Einstein

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter aims to introduce the developed dissertation project by offering details about its context, by explaining the problem that should be solved, as well as the research areas in which it is inserted and the objectives that should be accomplished by the end of the development. An overview of this report is also present in this chapter.

1.1 Context

The importance of text categorization is increasing due to the great amount of available information. Today, we face different challenges as a result of the amount of data that needs to be organized. For example, we are not able to categorize manually thousands of emails included in our email boxes, likewise, it is not possible to manually organize thousands of text documents used in data mining research works. Therefore, it is possible to infer that document/text data disorganization is as problem that should be resolved in order to maximize productivity in tasks that rely on large text data.

With recent developments in artificial intelligence, unsupervised learning and with better computer resources available, it is possible to automate tasks that would take an immense amount of time for the human to accomplish.

This project includes itself in the artificial intelligence and unsupervised learning areas because these are able to offer various algorithms and approaches that are capable of resolving the main objective of this work: organize text data into clusters.

This project was proposed by AMT-Consulting, a company that provides SAP consulting services. Therefore, this project may contribute to the improvement of certain services that could benefit from unsupervised learning approaches or from the automatization of text categorization.

1.2 Motivation and Objectives

As it was mentioned before, the process of organizing text information in categories is a very important task that will increase productivity in many areas and tasks that require text classification such as data mining projects, research works that require unlabeled data to be organized or email classification.

Consequently, this project aims to solve the problem of disorganized information in email inboxes present in AMT-Consulting company by creating a system that is able to categorize the email data retrieved from email inboxes in an efficient way and through the use of unsupervised learning algorithms and techniques. Since collaborators in this institution may waste several hours organizing their email inbox which is a very important tool in their work, an email categorization system could bring several benefits that may improve their productivity.

This objective will be accomplished through the use of different clustering techniques. Thus, the project is divided into three main parts.

The first one comprises the process of text data pre-processing and transformation where text will be cleaned and transformed in order to serve as input for the clustering algorithms. Secondly, several algorithms will be applied using the text data used in the previous step. Finally, evaluation metrics will be extracted from the algorithms in order to make conclusions about the best approach for the problem in this specific context.

More specifically, this project comprises the following objectives:

- **Create an email categorization system**: in order to create this system, several steps should be taken:
  - **Email extraction**: the email dataset given by AMT-Consulting comes in a specific *Outlook* file format, therefore, an extraction method is necessary to access the email information (subject, date, sender and body);
  - **Feature extraction**: in order the get the text data prepared to be fed into a clustering algorithm, a feature extraction method should be used so that the data may be transformed in a way that makes it usable by clustering algorithms;
  - **Clustering algorithm**: once the data is transformed in the feature extraction process, it will be used in a clustering algorithm that will be able to find patterns in data so that emails with different characteristics should be separated and those with similar ones should be in the same group;
  - **Topic retrieval**: Retrieve text data that identifies the groups that were created by the clustering algorithm;
- **Comparison between feature extraction methods**: several feature extraction methods will be used including *bag of words*, *tf-idf*, *word2vec* and *doc2vec*;
- **Comparison between different clustering algorithms**: three different clustering algorithms will be applied, each one belonging to a different category. It will be used K-means, belonging to partitional clustering algorithms, DBSCAN, that is included in density-based algorithms and agglometive clustering belonging to hierarchical clustering methods;

- **Choose the best methods**: the best feature selection and clustering algorithm will be chosen to integrate the email classification system. These methods, are chosen accordingly to the analysis of the results obtained through clustering metrics, execution times and advantages and disadvantages regarding the algorithm's nature.

There is a lot of work done in the text categorization area as we will see in Chapter 2, however, specific contexts, like AMT-Consulting inboxes, deserve a special approach since they contain datasets with unique characteristics due to the business context. For example, the emails may be written in more than one language or the inclusion of meeting dates may contain numbers that should not be taken into account in the topic extraction task.

In conclusion, the result of this project will be a system that relies on unsupervised learning methods to categorize text data retrieved from email inboxes and organize the text into folders. The names of the folders should represent the main theme or category of the classified emails.

1.3 Report Structure

This report includes four main chapters and the content of each one is explained bellow.

Chapter 2 describes the related work, including papers and other documents that report technologies and methodologies that influenced the development of this work. Consequently, it is included related work that regards feature extraction methods and clustering algorithms and their evaluation. Since this project also represents a study about unsupervised learning methods, several concepts are explained for a better understanding of the approach and implementation methods that were used.

Chapter 3 contains information on how the work was conducted in its different phases, including a summary of the approach used to achieve the main goals of this project and, lastly, all the details concerning the implementation regarding the methodologies and code used in each phase of the project, from data preparation to metrics retrieval.

In Chapter 4 it is included the experimental tasks that were made after the implementation phase. This chapter includes the experimental setup which indicates in which conditions the experiments were conducted, such as relevant information about the given datasets and their characteristics, used algorithms, which variables were used to analyze them and also, the results discussion that contains the main conclusions about the experiments that were conducted.

Finally, Chapter 5 contains the main conclusions of this project and the future work. These conclusions were drawn by comparing the project's results with the objectives established in Section 1.2. In the future work section, we can find details about what could be done to improve this project.

Chapter 2

Related Work

Concerning the text categorization problem, it is important to refer that most of the work done in this research area points to an unsupervised or a supervised approach [LY07]. The supervised method consists on classifying data using a model trained with labels, whereas the unsupervised method does not include any type of labeled data and tries to find patterns in the order to implement the classification process.

Another approach is the semi-supervised method that contains datasets with labeled and unlabeled data. It is able to categorize data by using techniques that retrieve the needed information from the labeled data that helps to identify properties of the unlabeled data [LY07] [GSGR17].

Since most of the text data present in AMT-Consulting dataset does not include sufficient labeled data in order to develop a supervised approach, this work inserts itself in the unsupervised learning field, meaning that the categorization task will be done through the process of finding patterns in the unclassified data in order to find similar characteristics in the dataset instances and create groups of data that will be considered folders.

This chapter includes sections about the related work regarding this project's main tasks, including feature extraction, clustering algorithms and clustering evaluation metrics as well as explanations about the most important concepts, approaches and algorithms used in this project.

2.1 Unsupervised learning

Unsupervised learning has been a very active area of research in the last few years because it gives us the possibility to obtain great learning results without trained data, which is a great advantage compared to the supervised learning approach where some labeled datasets may be very difficult to provide.

Ghahramani [Gha04] makes a comparison between reinforcement learning and unsupervised learning in order to prove how versatile unsupervised learning may be by explaining it from a statistical point of view. While reinforcement learning learns and interacts with the environment in order to receive rewards or punishments and learn the best ways to maximize the rewards, unsupervised learning has a different approach. Instead of having target outputs or interacting with the environment, it only receives inputs and, then, builds representations of this data in a way that is suitable for the task in question.

As we can see, the main goal of unsupervised learning is to find patterns in data. In this work we will use unsupervised learning to find text groups by clustering the given unlabeled dataset.

2.2 Data preparation

In order to divide the dataset into different clusters, we need to transform the text data into numerical data and feed it to the clustering algorithm. However, the data transformation and clustering algorithms are not responsible for data filtering, in other words, if it exists data that is not relevant for the clustering process, we need to remove it before any feature extraction task, otherwise, the irrelevant data will be considered in the clustering phase which will affect the clusters quality and, consequently, affect the overall categorization process.

In text clustering, there are several tasks that are commonly used to clean the text data [Sai15]. These pre-established tasks aim to reduce the data to its most relevant form so that the clusters may have the best quality possible. However, each dataset has its context and it may be important to adapt this tasks to the dataset that is being processed. In this project's case, all the inboxes exist in the company's context which means that there are some characteristics that will be common among these datasets.

The following operations are included in the text preparation process:

1. **Tokenize**: we call tokenization the task of dividing the text into tokens. This means that the string that contains the email text will be converted into an array where each element represents a token. This task should be the first one because once the text is tokenized, the other tasks will be easily handled with array methods.

2. **Remove punctuation**: all the punctuation should be removed from the text since it has no meaning in this specific context.

3. **Remove stopwords**: stopwords are the group of words that are very common but have no context meaning. Prepositions and adverbs (ex: the, for, your...) are a few examples. Therefore, this group of words should be removed in order to achieve better results.

4. **Remove non-context words**: non-context words represent the group of words that is not included in the stopwords but also does not have a relevant meaning for the problem. For example, in the problem of organizing the email boxes accordingly to the most frequent topics, the numbers included in the text should be considered non-context words, however, if we want to organize the inboxes by their date, the numbers become relevant for the categorization, therefore, they should not be considered non-context words.

5. **Stemming**: stemming is the process of converting the variations of a specific word to its root or stem. For example, the word "studying" should be converted to "study", however, the word "studies" would be converted to "studi" which is not a good form to use in topic extraction.

6. **Lemmatize**: the lemmatization process is highly correlated with the stemming process, however, its main difference is that it not only transforms the word into its stem but also takes into account the word context which is why this process is more complex. For example, considering the example above, the "studies" will be converted into "study" instead of "studi" which makes it a more reliable process for text pre-processing.

2.3 Feature Extraction

Most of the existing machine learning algorithms require numerical forms of data in order to achieve their goals. Consequently, there is the need to convert the initial format of the dataset into a numerical form, in other words, we need to extract the characteristics of the data and convert it into another format. This process is designated by feature extraction and it is important in several fields in the machine learning area such as image processing [KB14], facial recognition [Kwa08] and, most importantly in this context, text processing [Lew92]. The following subsections will explain the main feature extraction methods that were used in this work as well the related work regarding this methods. Among these approaches we can find the simpler ones such as *bag of words* and *tf-idf* weights and more complex and recent methods such as *word2vec* and *doc2vec*.

2.3.1 Bag of words

Bag of words is the process that converts text data into numerical data featuring the occurrence count of a certain word in each document.

This process works as it follows: firstly, it is created a vocabulary from the dataset that contains each word and then we will count the occurrences of each word in each document.

This process returns a matrix designated by term frequency matrix where one axis contains the words in the vocabulary and the other one contains all the documents, the intersection cell between the word and the document is the number of word occurrences in that specific document. The following simple example will explain the usage of this approach.

In this example the dataset is composed by three text documents:

- doc1: "let me explain the bag of words approach"
- doc2: "bag of words is a vectorization method"
- doc3: "bag of words is not the best vectorization method bag of words is a vectorization method"

Firstly, a vocabulary containing all the words is created. As we can see, the dataset has 24 words and the vocabulary contains 14 different words. If we store the vocabulary inside an array, the result will be:

vocabulary = [let, me, explain, the, bag, of, words, approach, is, a, vectorization, method, not, best]

The following step is to build a matrix that includes the frequency correlation between the words in the vocabulary and each document present in the dataset. The resulting matrix is represented in the Table 2.1.

Word	**Doc 1**	**Doc 2**	**Doc 3**
let	1	0	0
me	1	0	0
explain	1	0	0
the	1	0	1
bag	1	1	2
of	1	1	2
words	1	1	2
approach	1	0	0
is	0	1	2
a	0	1	1
vectorizing	0	1	2
method	0	1	2
not	0	0	1
best	0	0	1

Table 2.1: Bag of words matrix example

We are able to infer from the previous explanation that one of the *bag of words* advantages is its straightforward implementation and usage because it is a very simple and easy to understand approach.

However, this method includes several disadvantages like its sparsity and the fact that it does not take semantics into account.

The bag of words sparsity comes from the amount of features it may have, in other words, a dataset with a relatively small vocabulary may have an high amount of features with score values set to zero. Therefore, the models that use this approach will have to extract little information in an very large amount of features [Sl13]. On the other hand, as we saw in the example before, the algorithm does not take the words order into account which makes it an inadequate approach to use when the word semantics are an important characteristic in the project's context.

2.3.2 TF-IDF

Despite the fact that we could feed the *bag of words* result matrix into a clustering algorithm, those results would not be favorable because the matrix only takes into account the frequency of the word. Therefore, it would count words that have less meaning concerning the problem's context. For example, the word "is" has one of the highest counts in the vocabulary but it doesn't have any meaning in the problem's context.

The *tf-idf* measure aims to solve this problem because instead of only calculating the frequency of a word, *tf-idf* calculates a score for each word-document correlation by multiplying the term-frequency(tf) by the inverse document frequency(idf) [RU11]. These two measures are explained bellow:

- **Term frequency**: it is calculated by dividing the number of word occurrences in the document by the total number of words in the same document. In other words, it only counts the number of occurrences of a specific word in the document, however, it normalizes this value by dividing it by the number of words that the document contains (document length) in order to prevent that some words have higher score only because the document is larger, as we can see in Equation 2.1.

$$tf(w) = \frac{nr.\ w\ occurrences\ in\ doc}{nr.\ words\ in\ doc} \tag{2.1}$$

- **Inverse document frequency**: it is calculated by the *log* of the division between the number of documents and the number of documents that contain the specific word 2.2. While the term-frequency measure only counts the words occurrences making it a measure that considers the words equally important, the inverse document frequency measures how important the words are. By multiplying this measure by the term-frequency, the algorithm aims to deliver higher scores to rare words and minor scores to frequent words.

$$idf(w) = log(\frac{nr.\ docs}{nr.\ docs\ containing\ w}) \tag{2.2}$$

Therefore, we can summarize the *tf-idf* approach in Equation 2.3:

$$tfidf(w) = tf(w)\ *\ idf(w) \tag{2.3}$$

Tf-idf weights are very commonly used specially in text feature extraction task. Since its proposal [SB88], it suffered several variations in order the make the best of its advantages for each problem's context [Aiz03] [WLWK08].

2.3.3 Word2vec

Before explaining the *word2vec* technology, the concept of skip-gram model should be introduced. It was published by Tomas Mikolov [MCCD13] and this model tried do solve the atomic nature of several feature extraction methods, in other words, approaches such as *bag of words* and *tf-idf* only take into account the word count and do not construct any kind of relationship between the words. However, the main goal of skip-gram model is to create syntactic and semantic relationships between words in order to obtain a better representation of the text corpus in the feature extraction process.

Word2vec is a fairly recent technology published in 2013 by the very same author previously referred and several Google's researchers [MSC+13]. This method presents several improvements to the skip-gram model including the improvement of the feature vectors and the time it takes to train a skip-gram model.

The *word2vec* method works through the use of a neural network architecture that only includes three layers: input layer, hidden layer and output layer as it can be seen in the figure 2.1.

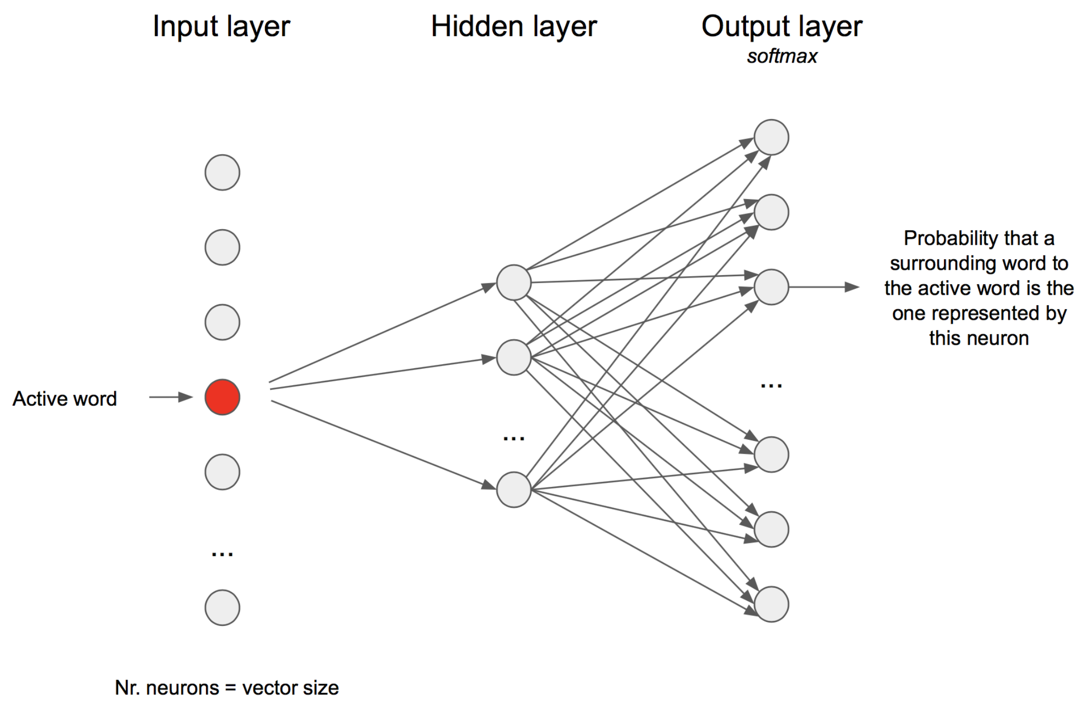
Input layer
Hidden layer
Output layer
softmax
Active word
...
Nr. neurons = vector size
Probability that a surrounding word to the active word is the one represented by this neuron

Figure 2.1: Word2vec Neural Network

The main objective of this neural network is to calculate the probability for each word in the vocabulary to belong in the surroundings of a specific word in a sentence. At the end of the training process, the weights of the hidden layer will constitute the feature vectors of the text.

Besides the fact that this approach is used in sentiment analysis problems due to its ability to detect context in the text [ZXSX15], this feature extraction method may also be implemented in a clustering process [Sie15] [MZ15].

2.3.4 Doc2vec

In 2014 Mikolov et al. [LM14] created the *doc2vec* approach based on *word2vec*. As the name indicates, instead of creating vectors for each word, it considers an entire document one vector. In its article, Mikolov describes its approach as "Paragraph vector" and it aims to overcome certain difficulties that the *bag of words* approach includes such as the lack of semantics in the feature representation.

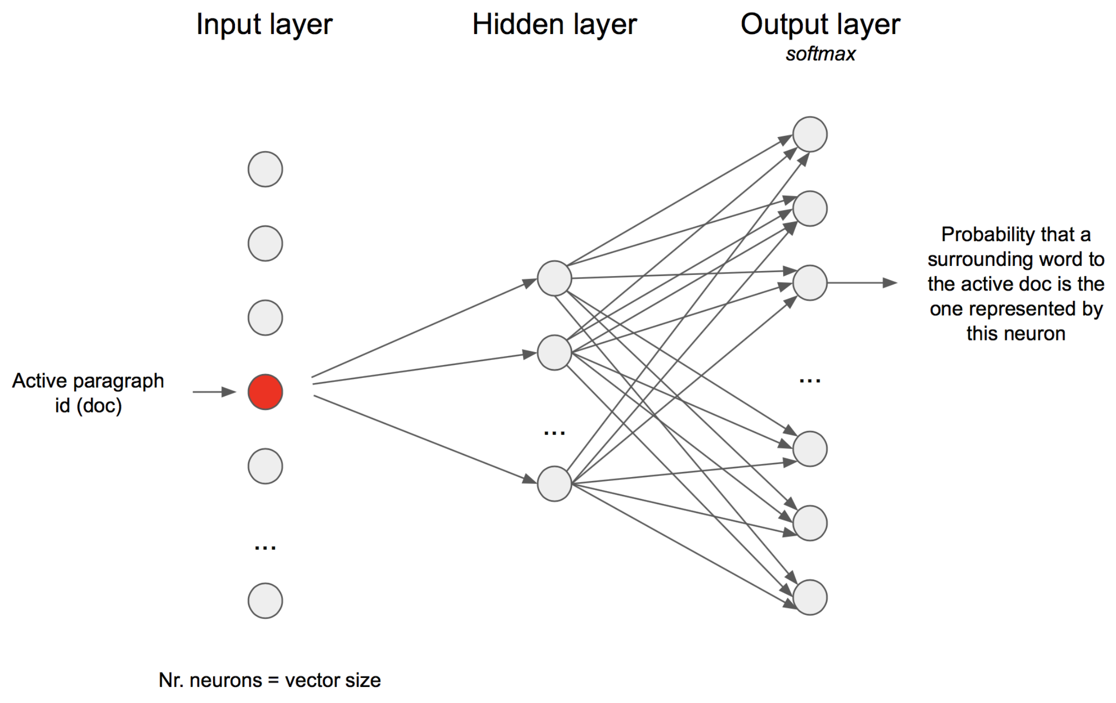
Input layer
Hidden layer
Output layer
softmax
Probability that a surrounding word to the active doc is the one represented by this neuron
Active paragraph id (doc)
...
Nr. neurons = vector size

Figure 2.2: Doc2vec Neural Network

2.4 Clustering

Clustering is a very important part of nowadays technological [SAJ17] tasks. Jain and Dubes [JD88] refer clustering as one of the fundamental parts of understanding and since clustering operates with data without pre-defined patterns, it inserts itself into the group of unsupervised learning methods.

Clustering is the process that divides data into groups where the instances that belong to same group are similar to each other and different from the instances belonging to the other groups [IPP16]. Therefore, the number of clusters goes from 1 which represents a group with all data, to the total number of instances in the data, which means that the larger the number of clusters, the more similar the objects inside each cluster are.

The clustering process needs a metric in order to classify if an object should be included in a cluster or not, in other words, it needs a measure of similarity to compare the different objects. In clustering, this measure is called "distance measure". Hence, we can say that the distance between two objects is the measure of how similar they are. This measure of similarity depends on the nature of the problem, however, the most common distance measures are "Euclidean distance" and "Cosine distance".

The previous distance measures can only work with numerical data in order to calculate the distance between two objects. This is why the feature extraction process (Section 3.3) is so important for clustering since it is responsible for the data transformation that allows the calculations of the distance measures.

There are several ways to calculate the distances in the clustering task but the ones that were used in this project were *euclidean* and *cosine* distances, which are explained in the following items.

- **Euclidean distance**: measures the distance between two points in the search space by measuring the line that connects both points. It is calculated by the square root of the sum of squares of the difference between each coordinates of the same dimension, for example, if the problem is in the two dimensional space and there are two points p_1 and p_2 with coordinates (x_1, y_1) and (x_2, y_2), the euclidean distance between these points will be equal calculated using Equation 2.4.

$$d(p_1, p_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.4}$$

  However, the previous formula may only be applied in two dimensional problems, in the text clustering field the number of dimensions is very high which makes it an n-dimensional problem. Therefore, the general formulation of the *euclidean* distance is presented in Equation 2.5, where n is the number of dimensions and i is the current calculated dimension.

$$d(p_1, p_2) = \sqrt{\sum_{i=1}^{n} (p_{2,i} - p_{1,i})^2} \tag{2.5}$$

- **Cosine distance**: instead of measuring the distance between two points in the search space like the euclidean distance, it measures the cosine *cos* between two vectors, for example, in the feature extraction phase of the clustering process, if the main goal is to transform the text data into vectors, the distance measure of two vectors present in the search space will be the cosine of the angle between them. Therefore, this distance is calculated by dividing the scalar product by the multiplied norms of the two vectors as it is shown in Equation 2.6, where A and B are two different vectors in the dimensional space and α is the angle between these two vectors.

$$d(A, B) = \cos(\alpha) = \frac{A \cdot B}{||A|| \, ||B||} \tag{2.6}$$

2.4.1 K-Means

Before explaining how the K-means algorithm works, it should be explained what is the partitional clustering category of algorithms. In this category the clustering algorithms create partitions in the data. These partitions will, then, constitute the clusters. These types of algorithms are good for datasets with great sizes but the necessity to previously explicit the number of clusters may be considered a disadvantage in certain contexts [JMF99].

K-means algorithm receives as input a set of points and the argument k which represents the number of clusters that the dataset will be split into. One of the main components of this algorithm are the centroids. A centroid represents an instance prototype of the cluster, in other words, it is not an instance but a prototype which should represent the center of the cluster.

Firstly, K-means places the centroids in random locations of the feature space. Then, it will repeat the following process until it finds convergence (when the centroids stop updating their positions between iterations):

1. for each point p_i in the dataset D
   (a) find the nearest centroid to p_i
   (b) assign p_i to the cluster C_j
2. for each cluster C_j
   (a) calculate the mean of all the points p_i belonging to C_j in the previous step
   (b) update C_j centroid by assigning the mean value calculated

In a more technical explanation, the main objective of this algorithm is to minimize the function that calculates the "sum of squared errors" that should be calculated using Equation 2.7 where x_i represents the set of instances to be clustered, C_i indicates the generated k clusters and μ represents the mean of the points included in C_i [ZWM14].

$$SSE(C) = \sum_{i=1}^{k} \sum_{x_j \in C_i} \|x_j - \mu_i\|^2 \tag{2.7}$$

Since K-means is one of the most commonly used clustering algorithms, there is already research work regarding the task of text clustering.

Even today, K-means and its variations are widely used in applications and research areas that have the necessity to organize text into clusters.

Shetty et al. [SK17] presents a similar approach to this dissertation because it includes the whole clustering process despite the different final goal. In this paper the documents are also clustered into groups, however, after this task, several sentences are retrieved in order to make a summary of the inserted documents, whereas in this dissertation's project, instead of sentences, topics will be retrieved in order to characterize the clusters. It is also similar in the fact that it includes several methods and approaches that will be used in this work, such as text pre-processing tasks, feature extraction with *tf-idf* and K-means as the clustering algorithm. A similar clustering approach [AAU$^+$17] was also used for different languages.

Other work related with the K-means algorithm aims to optimize its performance by using other methods such as the Latent semantic analysis (LSI) [ASFA17] or other approaches such as categorizing text using a recursive version of K-means that stops when the desired clusters are obtained [GSGR17] or an improved version of the algorithm [XHLL16] that mitigates its high dependency on the initial clusters centers that translates in better recall and precision rates for text clustering or Chen et al. [CYTZ09] that used weighten K-means in attribute conversion and *euclidean* distance to perform the text clustering analysis.

Since its creation, K-means was not only used in several clustering applications but it also suffered several mutations by researchers with the goal of improving its performance. One of this examples, was the K-means variation [AV07] that will be used in this work, K-means++. The difference between this variation and the original algorithm lies in the position of the centroids. In the original K-means, these positions are chosen randomly, however, in K-means++ it is used a randomized seeding technique that should be able to increase the performance of the algorithm. This implementation is available in the *scikit-learn* python library.

2.4.2 DBSCAN

DBSCAN (density-based spatial clustering of applications with noise) was created by Ester et al. [EKSX96] with the intent to cluster data containing noise. This algorithm belongs to the density-based category of clustering algorithms. Density-based algorithms try to find clusters in the most dense areas of the feature space instead of dividing the space into partition zones like partitional algorithms such as K-means do. Therefore, DBSCAN aims to return clusters where each cluster corresponds to an high density region in the search space.

DBSCAN receives as input the set of points from the dataset, the Neighborhood parameter (N) and the *min_points* parameter. While N defines the diameter that surrounds a point in the dataset, the *min_points* parameter is the minimum number of points that is necessary to consider a certain instance a *core point*. Figure 2.3 represents the two main parameters of the algorithm as well as the points attributions that will be explained next.

In this algorithm's execution, each point in the dataset will be labeled accordingly with three different labels: *core*, *border* and *noise*. A *core* point is a point that includes at least the value of *min_points* in its neighborhood N, a *border* point is a point that has at least one *core* point in its neighborhood N and a *noise* point is a point that is neither a *core* or a *border* (Figure 2.3).

The algorithm works by assigning each *core* and *border* points into a cluster. Then, it chooses a point in the feature space and performs a depth-first search. This process will be repeated until all core points points are assign to a cluster.

DBSCAN has some benefits such as the fact that it is able to discover any number of clusters, in other words, it is able to find the clusters without the previous k number and most importantly, it can detect and ignore outliers. Nevertheless, one of its major drawbacks is the fact that it is very sensitive to the parameter N. When N value is too low, there is the possibility of sparse clusters be considered as noise and if N is too large, two different clusters may be merged together. Therefore, the N must be chosen very carefully in order to obtain the best results.

Due to its popularity, specially in the category of density-based algorithms, DBSCAN was a targeted algorithm for many research work. Just like K-means, several variations of the algorithm were introduced in order to improve its performance [KRA$^+$14].

For example, a fuzzy version of this algorithm was introduced [KKR15] so that the algorithm could be able to group data in a fuzzy manner which means that there is a certain probability for

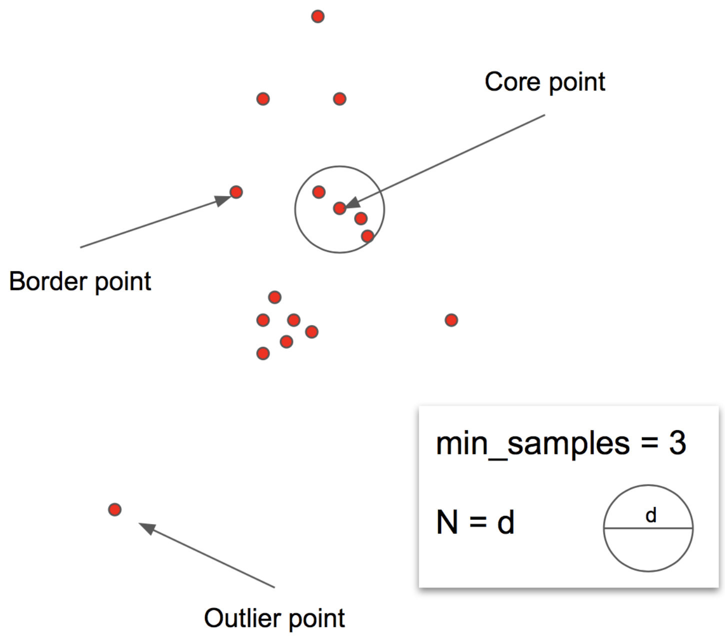

Figure 2.3: DBSCAN point assignment

one specific instance to belong in a certain group contradicting non-fuzzy algorithms such as K-means where a certain instance is obligated to belong into one group only. A parallel version of the algorithm was also created by Götz et al. [GBR15] that should be useful for vary large datasets or clustering problems that have a very high load in computational resources.

DBSCAN is also present in recent clustering applications such as sentence summarization [RK17] or in more specific tasks such as URL clustering [MBMM17].

2.4.3 Agglomerative clustering

Agglomerative clustering is a technique that belongs to a very different clustering category comparing to the two previous ones. It is an hierarchical clustering algorithm. There are two ways of applying this category of clustering algorithms: one where we begin with only one cluster that contains all the points in the search space and we split it until each point in dataset is a cluster (top-down approach) and we also may apply the second approach where we start with the number of clusters equal to the number of points in the dataset and we connect them until we reach a point where we only have one cluster that contains all the points in the search space (bottom-up approach). Agglomerative clustering applies the later approach.

At the end of the hierarchical clustering algorithm we get an hierarchical scheme that may be represented by a structure called dendogram. A dendogram is a tree like structure in which the root is the cluster that contains all the instances of the dataset and the leaves are the all the instances of the dataset.

The agglomerative clustering algorithm may be specially useful for data discovery not only because it can generate structures like dendograms but also because it is a sequential process where the merging clusters will give useful information about the data [SB13].

Agglomerative clustering algorithm works with three simple steps:

1. Create a collection of clusters C (length of C equals the number of instances in the dataset)
2. Repeat until there is only one cluster
   (a) find two clusters that are the closest (c_1 and c_2)
   (b) merge these two clusters together creating a cluster $c_{1,2}$
   (c) add cluster $c_{1,2}$ to the collection C
   (d) delete c_1 and c_2 from the collection C

One of the most important components of this algorithm is the criteria used in the process of merging the clusters. Since the similarity distances (such as *euclidean* or *cosine*) are only applicable in instances of the feature space, we have to use other methods in order to calculate the distance between the clusters because, in agglomerative clustering, we have to merge the clusters that have the least distance between them first, in other words, for a certain cluster c_1, we have to calculate which is the closest cluster c_2 and merge them together.

Therefore, the four most used techniques to calculate the cluster distance (also known as linkage criteria) are the single linkage, complete linkage, average linkage, and ward linkage [MC11]. Considering two sets of observations A containing instances a_i and B containing instances b_i, consider the following [Sha09] [Uni]:

- **Single linkage**: the distance between the clusters will be equal to the similarity distance between the two closest instances belonging to each cluster A and B (Equation 2.8).

$$d(A,B) = min\{d(a_i,b_i)\} \tag{2.8}$$

- **Complete linkage**: it is the opposite of single linkage. The distance between the two clusters will be equal to the similarity distance between the farthest instances belonging to each cluster (Equation 2.9).

$$d(A,B) = max\{d(a_i,b_i)\} \tag{2.9}$$

- **Average linkage**: it is a more complex approach since it has to calculate the distance between all the instances inside the clusters. Then, the final distance result between the clusters will be the calculated average between all the pairwise distances as Equation 2.10 indicates where k and l equal the sizes of A and B respectively.

$$d(A,B) = \frac{1}{kl}\sum_{i=0}^{k}\sum_{j=0}^{l}\{d(a_i,b_j)\} \tag{2.10}$$

- **Ward linkage**: is it considered a variance-minimizing approach that calculates the the sum of squared differences within the clusters. It is similar to K-means objective function. Equation 2.11 represents the cost of merging two clusters which should be minimized where m indicates the cluster center.

$$merging\_cost = \sum_{i \in A \cup B} \|x_i - m_{A \cup B}\|^2 - \sum_{i \in A} \|x_i - m_A\|^2 - \sum_{i \in B} \|x_i - m_B\|^2 \tag{2.11}$$

Just like K-means and DBSCAN, several recent studies have been published using the agglomerative clustering technique applied to text clustering.

To examplify, agglomerative clustering was applied in the text summarization field [SSAV17] by using a sentence ranking approach and also in topic detection [LL16].

2.4.4 Evaluation

One of the most noticeable differences between unsupervised and supervised learning lies on the validation process. In supervised learning we immediately have access to the correct information in order to make a comparison on how the model's results distance themselves from the correct results and, therefore, we are able to make an assertion on how good the model is. However, in unsupervised learning, there are not any previous labels that will help to make the comparison between the results obtained and the results that the model is supposed to create.

In unsupervised learning, more specifically, clustering, there are two main ways to validate the model. If we are able to find what are the results that the model should obtain, we can borrow some of the concepts used in supervised learning. These approaches are designated by external measures because we are evaluating the model with external information. However, normally, when we use unsupervised learning techniques, we do not have that kind of information available, we only want to find patterns in the data. This way, we have to use metrics that will not depend on external data and that are able to evaluate the quality of the model. In the clustering case, the quality of the partitions (clusters) created in terms of compactness and separation.

The validation metric used in this project was internal and it is designated by *silhouette* coefficient.

The *silhouette* coefficient is able to measure how good the clusters are defined [Rou87] and it may be calculated through two measures as explained in the *scikit-learn* library documentation [PVG⁺11]: the mean distance between an instance and all the other ones in the same cluster (a) and the mean distance between an instance and all the others in the closest cluster (b) [iW03]. Therefore, the *silhouette* coefficient may be expressed in Equation 2.12.

$$s = \frac{b - a}{max(a, b)} \tag{2.12}$$

By calculating these two values (a and b), the metric is able to evaluate the compactness and the separation of the model.

The score calculated by this measure may vary between -1 and 1. When the value is close to -1 means that the model is incorrect and the clusters are mixed while values close to 1 indicate that the clusters are well separated.

2.5 Email classification

Since the dataset provided to develop this work consists of emails retrieved from inboxes, it is important to do a research of previous work related with email classification. Email classification is referred when we want to insert an email into a category. These categories may be related to the email's topic or the email's characteristics such as classifying it as spam or not.

The problem of email classification has been explored for more that two decades, that means, since the quantity of email information was so big it became necessary an automatic way to organize it. Most of these works used the Enron dataset [KY04], which has become the number one benchmark for email classification.

Bekkerman [BMH04] presented a study about the email organization into folders by using the dataset above referred as benchmark. In this experiment, emails were organized accordingly to different measures other than the content, such as the email timeline.

Tang et al. [TPL14] offer a general view about the tasks related with email organization. It includes different operations related with emails such as contact analysis, email network, data representation, text pre-processing, spam detection and most importantly: email categorization.

However, the previous two papers only offer a supervised approach for email classification. Despite the fact that the supervised approach is not considered in this dissertation, the text preparation phase is very similar.

Alsmadi et al. [AA15] made a comparison between the supervised and unsupervised learning approaches. In the unsupervised phase it was used a *tf-idf* approach and the clustering algorithm K-means.

Proceeding to the unsupervised approach, Patil et al. [PD15] used a common approach related with clustering methods where the documents suffered a pre-processing phase, a term-frequency matrix was calculated and K-means algorithm was applied. However, between the matrix and K-means, a new similarity matrix was created related with the email's context, and then, this similarity measure was compared to the other common ones such as *euclidean* and *cosine* distances.

Chapter 3

Approach

This chapter aims to describe the details about the project's development. It is divided into six sections that describe each stage of the project. Firstly, in Section 3.1, there is an explanation about the extraction of the emails from that email boxes, in other words, the process of transforming the initial *.pst* email file into extracted email data. Secondly, the steps regarding the dataset are described in Section 3.2 and while this section describes the dataset in a general view, Section 3.3 aims to describe the steps regarding the dataset in a transformation level, in other words, how to prepare data so that it can be fed into the clustering algorithms. Thirdly, Section 3.4 includes the used clustering algorithms as well as the way they were implemented. Section 3.5 includes some of the validation metrics that will be used to analyze the project's results and Section 3.6 includes the final stages of the classification system's implementation.

As mentioned before, the main objective of this work was the development of a system that is able to classify different emails and organize them into folders.

An outlook file must be given to the system (preferably an inbox retrieved from the company AMT-Consulting) so that it can process the file and, consequently, create a directory with all the organized folders and emails. The main steps of the system are represented in Figure 3.1.

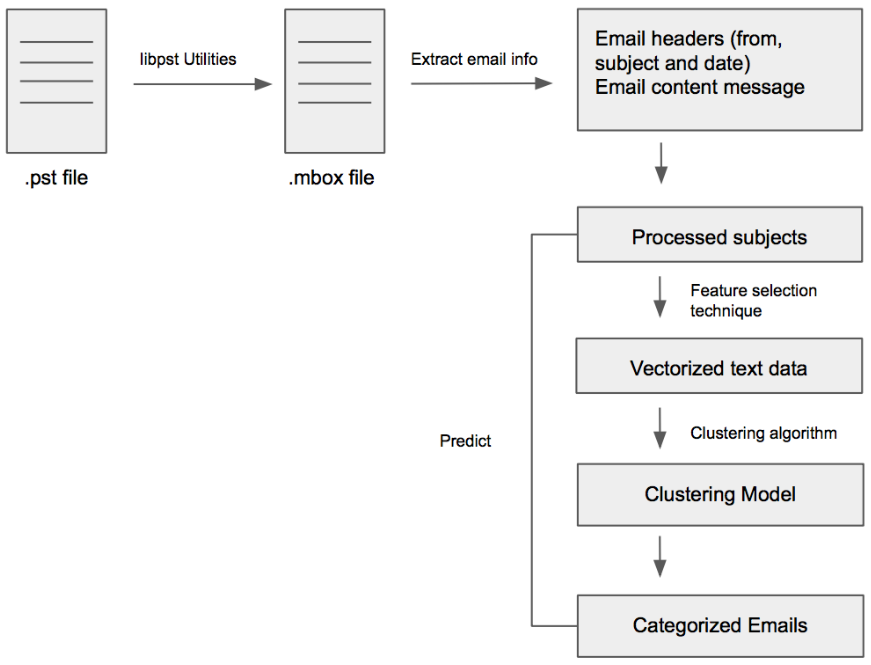

Figure 3.1: System main steps

3.1 Email extraction

Initially, the system receives an Outlook data file (.pst) as input. A file with the *.pst* extension packs all the information about an Outlook email account, including email messages, contacts, among others [Mic]. However, this email format is not very common and, consequently, there are not any viable libraries or frameworks able to extract the email messages directly from this file type. Therefore, the file should be converted into a more functional format like *mbox* that is used by Apple and Google email systems. This file also stores the email account content like the *pst* file but there are more API options to extract the email content from this file format.

The python library *libpst utilities* was used to convert the *.pst* files into *.mbox* format. This library works in the command line and among other features, it enables the conversion from *.pst* files to *.mbox* files. Since the library only works in the command line, the best approach was to make system calls with python scripts with the use of the python libraries *os* and *subprocess* as it can be seen in Listing 3.1. Consequently, by the end of this step, the *mbox* file is created and stored in the same directory as the project.

```python
import os
import subprocess

inbox_number = 1

if not (os.path.isdir(folder_mboxes)):
    path_to_mboxes = ’email_mboxes/inbox’ + str(inbox_number)
    os.makedirs(path_to_mboxes)
    subprocess.check_call("readpst -o " + path_to_mboxes + " inboxes/inbox" + str(
        inbox_number) + ".pst", shell=True)
```

Listing 3.1: .pst to .mbox convertion

As it can be seen in Listing 3.1, if the folder containing the *mbox* files does not exist, which means that the conversion process was not executed for this specific email inbox, then, a process is created in order to make a command line call that reads the *pst* file and extracts an *mbox* file for each folder that the email inbox contains. In this command line call, the command *readpst* was used, which is one of the conversion possibilities for the *pst* file. The flag *-o* gives the possibility to specify the path to the folder in which *mbox* files will be stored and the last parameter represents the name of the *pst* file that should be converted [Smi].

In the next phase, the email messages content must be extracted from the *mbox* file. In order to achieve this, two python libraries were used: *mailbox* and *email*. The *mailbox* library enabled us to store the *mbox* file content in a python's object variable (line 6 of Listing 3.2) and then, the *email* library was used to extract the several parts of the email from the python variable where the *mbox* content was stored. These extracted components are the sender of the message, the date and time in which the message was sent, the subject of the email message (if there are any messages without subject, the subject is converted to "nosubject") and lastly, the email body. The python code that was used to extract this email components is present in Listing 3.3.

```python
1    import mailbox
2
3    with open(emails_processed_destination + ".txt", "w+") as f:
4          total_email_counter = 0
5          for mbox_name in mbox_list:
6              mbox = mailbox.mbox(folder_mboxes + "/" + mbox_name)
7              total_email_counter += list_mbox_content(mbox_name, mbox, f)
8          print("Total number of emails: " + str(total_email_counter))
```

Listing 3.2: Mbox python object

```python
import email

def extract_mbox_content(mbox_file):
    ...
    for message in mbox_file:
        message_date = message['date']
        if message_date == "" or message_date is None:
            message_date = "nodate"
        try:
            message_subject = str(email.header.make_header(email.header.
                decode_header(message['subject'])))
        except UnicodeDecodeError:
            message_subject = "nosubject"
        if not message_subject:
            message_subject = "nosubject"
        message_without_punctuation = strip_punctuation(message_subject)
        message_from = str(email.header.make_header(email.header.decode_header(
            message['from'])))
        message_body = str(getBody(message))
    ...
```

Listing 3.3: Mbox content extraction

Listing 3.2 and Listing 3.3 aim to explicit the final stages of the email extraction phase.

In Listing 3.2, it was used the *mailbox* library to extract the *mbox* content (one inbox may have several *mbox* files) and store it into a python variable as it is explicit in the line 6.

However, Listing 3.3 is more complex because it takes a few more steps. This code iterates through the *mbox* object and for each message contained in this file, it extracts and decode the subject, date, from, and the body.

Once this process is concluded, the extracted emails will be stored in a folder correspondent to a certain inbox. The folder contains text files with the extracted email data where the title is a concatenated string joining the subject and the date. This title constitutes a safety measure because different emails may contain the same subject and by joining the date to its subject, it prevents overlapping emails. The content of these text files includes the sender, the subject, the date and the text portion of the email body.

However, only a portion of the information above referred will be used in the clustering phase. As it was mentioned before, only the subjects will be used in the clustering task due to the non-relevant information that is present in the emails body in the AMT-Consulting dataset. Therefore, this data must be separated from the text files above referred. While the email data is being extracted and stored into the email files, when the subject is extracted, it will not only serve as the title for the text email file but it will also be pre-processed and stored in another text file. In a better explanation, for each inbox, it will be created a text file that, in each line, will contain the subject of the email after the pre-processing tasks which are explained in the following Section 3.2. Figure 3.2 explains schematically the referred extraction process.

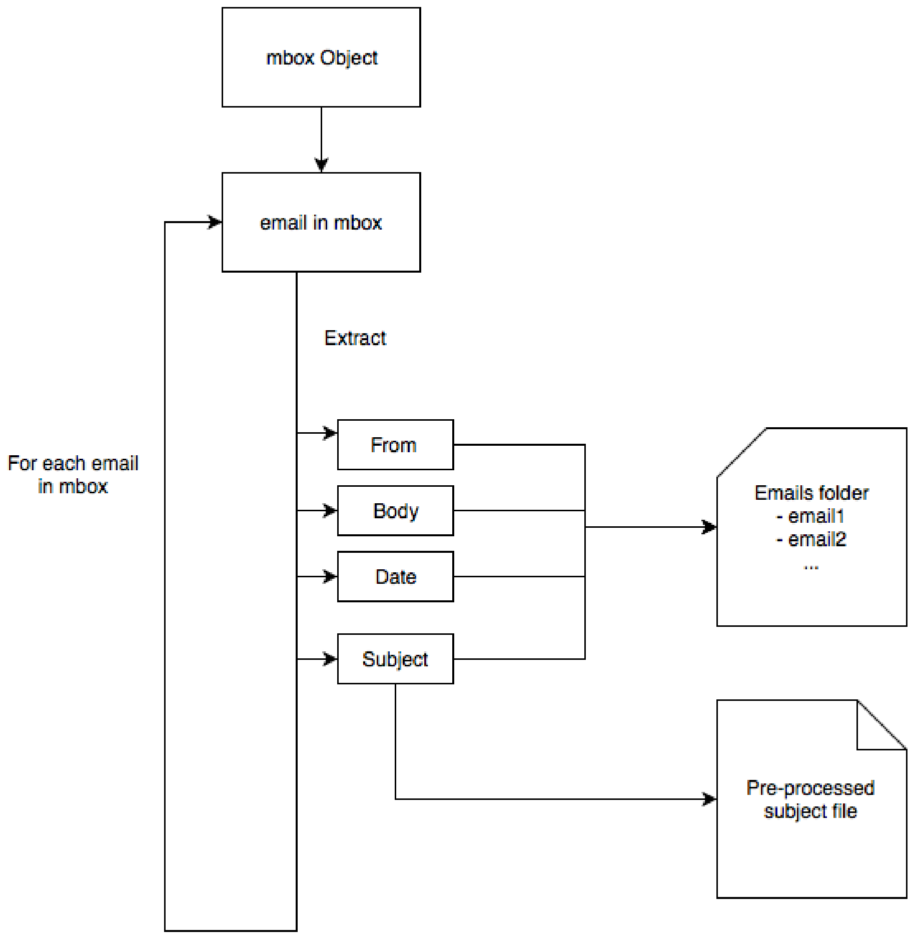

Figure 3.2: Extraction process

3.2 Data Preparation

In the section above, a pre-processing module for the email subjects was referred. This module's primary objective is to clean the subjects data in order to be prepared to be used in a feature extraction technique and, consequently, in a clustering algorithm.

This cleaning process contains the main techniques referred in Section 2.2. Elements such as punctuation, stopwords, email prefixes, context words and emojis were removed and the module returns the cleaned subject which is stored in the text file that contains all the processed subjects correspondent to a specific inbox. However, in this data preparation context, it was not possible to apply stemming or lemmatization techniques due to the multi-lingual nature of the dataset.

On the other hand, since this is a system that will classify emails from different email boxes there are certain drawbacks, for example, a certain email inbox may have different characteristics that will not be able to be treated. In other words, since we do not have a specific dataset that may be treated with all the preparation details, we have to rely on the general data preparation techniques and details of the problem's context. One example, are the context words present in emails about meetings and daytimes. This is a detail that may be removed from the email corpus since it does not provide any important information about the email category.

The diagram in Figure 3.3 contains all the pre-processing tasks that were made to clean the email subjects.

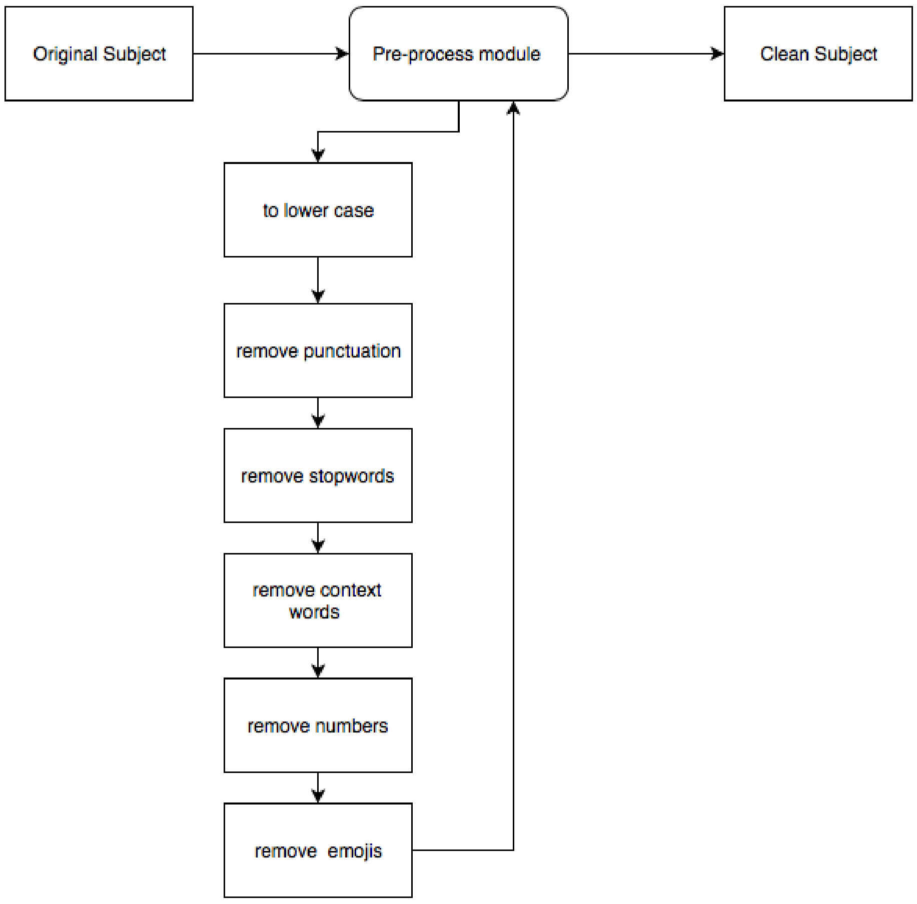

Figure 3.3: Pre-processing module

3.3 Feature Extraction

The next step in the clustering process is the extraction of features present in the dataset. In order to feed the text data into the clustering algorithm there is the need to make a conversion from text data to numerical data, in other words, the string values that represent the pre-processed subjects of the emails should be converted into a numerical matrix that should maintain the original characteristics.

As it was mentioned in Chapter 2, the methods of feature extraction used in this project were *bag of words*, *tf-idf*, *word2vec* and *doc2vec* and two python libraries were used to implement these methods: *scikit-learn* and *gensim*.

The *scikit-learn* library includes a *bag of words* implementation with the method *CountVectorizer* that creates a sparse matrix with word occurrences in each document. On the other hand, the method *TfIdfVectorizer*, that is included in the same python library, extracts the *tf-idf* representation by firstly appyling *bag of words* approach and then converts the matrix of word occurrences to a matrix with *tf-idf* features.

In order to implement the *word2vec* and *doc2vec*, the *gensim* library was the best option because it includes all the necessary methods to easily implement these feature extraction methods. In *gensim* library, are included not only the implementations of these feature extraction methods but also methods that convert the data to analyze into the right format in order to apply *word2vec* and *doc2vec*. The fact the this library includes a well documented API [Gen] also represented a great advantage.

Word2vec and *doc2vec* are techniques that require a training process. Some pre-trained models are already available so that they can be used in different problems, however, a model that fits this problem's context was not found due to the inconsistent nature of the used text data (text with different languages). Therefore, these techniques were trained using the subjects contained in the email inboxes.

3.4 Clustering process

Once the feature extraction phase is completed, the data is ready to be fed into the clustering algorithms, in other words, from the feature extraction process we obtained the data in a matrix form ready to serve as input for the clustering algorithms.

In this project's phase, three types of clustering algorithms were used in order to achieve a comparison between them and, for each type of clustering, an algorithm from that category was chosen. These three categories were partitional, density-based and hierarchical clustering and the implemented algorithms were K-means, DBSCAN and agglomerative clustering, respectively.

The implementation of these algorithms was made with the *scikit-learn* library and some aspects are similar throughout the different algorithms. Firstly, we declared the library's right method for the desired algorithm and fill out the necessary parameters (these may vary accordingly with the algorithm that's being used) and assign it to a python object variable that will store the model. Secondly, we use the matrix that was generated in the feature extraction models and associate it with the object's variable by calling the *fit* method and use the matrix as a parameter. Therefore, these are the general steps to take in performing a clustering task using the *scikit-learn* library. As it was mentioned before, several aspects referring each implementation may vary and they will be explained in the following subsections.

3.4.1 K-means

As it was mentioned in Chapter 2, Section 2.4.1, K-means is arguably the most understandable and most commonly used clustering algorithm. Therefore, it makes sense to be the first algorithm to implement.

The used K-means implementation was included in the *scikit-learn* python library. It not only includes the common arguments such as the number of clusters that should be inserted and the max number of iterations, but most importantly, it includes the option for the K-means "init" method as argument. With the "init" parameter, the algorithm may create the initial centroids positions in random locations or optimize these initial positions in order to achieve a faster convergence (K-means++). Listing 3.4 provides one of the functions created to run the K-means algorithm.

```python
from sklearn.cluster import KMeans

def apply_kmeans(vectorized_data, nr_clusters):
    print("Run K-means model with: " + str(nr_clusters) + " clusters")
    applied_model = KMeans(n_clusters=nr_clusters, init='k-means++', max_iter=100,
        n_init=1)
    applied_model.fit(vectorized_data)
    return applied_model
```

Listing 3.4: Example of *scikit-learn* K-means method

In the example explicit in Listing 3.4, we can find the created python method that receives as parameters the data that was returned by the feature extraction approach and the number of clusters that the algorithm will include. As for the K-means call, besides the number of clusters and the "init" method that were explained above, it contains the parameter "max_iter" that represents the maximum number of iterations for a single K-means run and it also contains the "n_init" parameter that gives the possibility to run more instances of the K-means algorithm with different centroid seeds.

3.4.2 DBSCAN

The density-based algorithm implementation, DBSCAN, was very similar to K-means. In this approach, the model variable was created by calling the DBSCAN method in the *scikit-learn* library with the most important parameters for the correct implementation of the algorithm. As it was explained in Section 2.4.2, these parameters are *N* (neighborhood) which represents the distance between two samples in order to consider them in the same neighborhood and *min_samples*, which represents the maximum number of instances present in the same *N* in order for this to be considered a *core* point. Once the model variable is created, the rest of the implementation is similar to the steps referred above as it can be seen in Listing 3.5.

```python
from sklearn.cluster import DBSCAN

def apply_dbscan(vectorized_data, n, min_points):
    print("Run DBSCAN model")
    model = DBSCAN(eps=n, min_samples=min_points)
    model.fit(vectorized_data)
    return applied_model
```

Listing 3.5: Example of *scikit-learn* DBSCAN method

3.4.3 Agglomerative clustering

The *scikit-learn* implementation of the agglomerative clustering algorithm includes several parameter options that will drastically change the way it behaves. As it was mentioned in Section 2.4.3, the agglomerative clustering algorithm works by merging similar instances together in order to create clusters and then, merge these clusters.

However, the merging task needs a linkage criteria and this is one of the most important parameters to tune the agglomerative clustering implementation. In this project, three different linkage methods were used in order to retrieve metrics for comparison: *ward*, *complete* and *average*. These linkage methods could be set in the linkage parameter of the agglomerative clustering method.

Secondly, affinity parameter is also an important one since it corresponds to distance metric used to compare the instances inside the clusters whose linkage will be computed by the norm specified in the linkage parameter.

Lastly, there is the parameter called "n_clusters" which indicates the maximum number of clusters to use. This parameter has a different meaning from the one present in the *scikit-learn* K-means method, because while in K-means it defines the number of centroids to use and, consequently, the number of clusters that will be in the algorithm's output, the parameter "n_clusters" in agglomerative clustering is the maximum number of clusters that will be obtained. In other words, since this algorithm works by merging the clusters, this parameter will set the maximum number of merged clusters. Therefore, once the algorithm achieves the "n_clusters" count, it stops its execution.

```python
from sklearn.cluster import AgglomerativeClustering

def apply_agglomerative_clustering(vectorized_data):
    print("Run agglomerative clustering model")
    model = AgglomerativeClustering(n_clusters=15, affinity="euclidean", linkage="
        ward")
    model.fit(vectorized_data)
    return model
```

Listing 3.6: Example of *scikit-learn* Agglomerative clustering method

As we can infer from Listing 3.6, the agglomerative clustering implementation is also very straightforward because it takes the three main arguments for the algorithm to run: the maximum number of clusters (n_clusters), the affinity method (affinity) which can be, for example, "euclidean" or "cosine" and, finally, the parameter "linkage" that represents the clusters linkage method, for example, "ward", "complete" or "average".

3.5 Model evaluation

In order to evaluate clustering models, *scikit-learn* provides several methods that include internal and external clustering evaluation metrics. However, since we do not have pre-defined labels, we will use only the *silhouette* score to measure the clustering model quality and the algorithm's execution time to measure its performance.

3.5.1 Silhouette index

Firstly, we are able to calculate the *silhouette* score by using the method "silhouette_score" which receives the parameters: "X", "labels" and "metric". The parameter "X" represents the dataset features that were extracted in the feature extraction phase, "labels" is the output of the clustering algorithm, in other words, the labels indicate the connection between an instance and the created clusters and the parameter "metric" corresponds to the distance metric used to calculate the distance between the dataset samples. Lastly, the method returns the mean of all instances' *silhouette* index.

3.5.2 Execution time

The execution time was measured using python's *time* library which is able to return the current time in seconds. Therefore, if we get the current time before and after the algorithm's execution, we are able to obtain the algorithm's execution time. In Figure 3.7 there is an example on how the execution time was measured.

```python
from sklearn.cluster import AgglomerativeClustering
import time

def agglomerative_clustering_time(vectorized_data):
    print("Run agglomerative clustering model")
    start = time.time()
    model = AgglomerativeClustering(n_clusters=15, affinity="euclidean", linkage="
        ward")
    model.fit(vectorized_data)
    finish = time.time()
    execution_time = finish - start
    return execution_time
```

Listing 3.7: Example on how to measure the algorithm's execution time.

3.6 Prediction and Foldering

Once all the models are created and evaluated, the next step, is the email classification.

In this classification, the system receives an email as input and it classifies the email accordingly with the words contained in the its subject. To put it differently, the email's subject will be extracted and transformed and, afterwards, it will serve as input for the clustering model which will associate this subject to the cluster with the most similar characteristics.

As it was referred in Section 3.1, during the extraction phase, a folder including the email's content with a title that results from the process of string concatenation between the subject and the date. However, in order to easily extract the subject for classification after the clustering model is completed, the subject is connected to the date using a unique key. The reason for this is that it needs to be a sequence of characters that is very unlikely to appear in the email's subjects. This process, enables us to loop through the email folder and retrieve all the email subjects for classification.

In order to implement this task, *scikit-learn*'s *predict* method can be called from the variable's object that contains the model and it receives as parameter, the element that should be classified. However, this element should be properly processed, otherwise, the prediction will not be correct. In this project's context, once the subject is retrieved, it must be pre-processed before it serves as input for the *predict* method, which includes the retrieval of all the stop words, context words, punctuation etc. In summary, the subject to classify should suffer the same pre-processing transformations as the data that was used to create the clustering model.

The code presented in Listing 3.8 is responsible for the classification process, in other words, for taking the subjects and classify them into folders using the created clustering model.

```python
import os
import subprocess
from pre_process import pre_process_text


categorized_emails_folder = "categorized"


def load_email_subjects(inbox_to_process):
    return os.listdir(inbox_to_process)


def process_subjects(subjects):
    email_and_processed = []
    for subject_date in subjects:
        subject = subject_date.partition(".!!..!!.")[0]
        email_and_processed.append([subject_date, pre_process_text(subject)])
    return email_and_processed


```

```python
def save_emails_to_folders(inbox_to_process, vectorizer, clustering_model, nr_clusters):

    path_to_inbox = "emails/inbox" + str(inbox_to_process)
    subjects = load_email_subjects(path_to_inbox)
    original_and_processed_emails = process_subjects(subjects)

    terms = vectorizer.get_feature_names()
    order_centroids = clustering_model.cluster_centers_.argsort()[:, ::-1]

    if not (os.path.isdir(categorized_emails_folder)):
        subprocess.check_call("mkdir " + categorized_emails_folder, shell=True)

    # hashmap: for each cluster i the top terms are assign
    email_folders = {}

    print("Creating folders")  # populate hasmap
    for i in range(nr_clusters):
        top_terms = ""
        for ind in order_centroids[i, :8]:
            top_terms += terms[ind] + "_"
        email_folders[i] = top_terms
        subprocess.check_call("mkdir -p " + categorized_emails_folder + "/" + top_terms, shell=True)

    print("Assigning emails to folders")  # use hasmap
    for subject in original_and_processed_emails:
        Y = vectorizer.transform([subject[1]])
        prediction = clustering_model.predict(Y)
        folder = email_folders[prediction[0]]
        subprocess.check_call("cp" + " " + "'emails/" + "inbox" + str(inbox_to_process) + "/"
                              + subject[0] + "'" + " " + "'categorized/" + folder + "'", shell=True)
    print("Foldering process finished")
```

Listing 3.8: Python code used in the folder organization.

As we can infer through the analysis of the previous python code, the main structure lies on the *save_emails_to_folders* method. This method receives as parameters the inbox that contains the emails that should be classified, the data received from the feature extraction method *vectorizer*, the clustering model to be used in the classification process *clustering_model* and the number of clusters/folders *nr_clusters*.

As this method begins its execution, it generates the folder path that contains the emails that should be classified accordingly with the parameter *inbox_to_process*. Then, the email subjects will be retrieved from the email folder and pre-processed.

Once this process is done, an hashmap will be created where the keys are the cluster/folder number and the value is the top terms the were found in that specific cluster. This is followed by the creation of the folders through the use of the command *mkdir*. Consequently, the names of the folders will be the top word terms present in a cluster so that each folder may be characterized in an easy way.

The final step includes the actual classification of each subject. In this phase, the folders are already created and named, consequently, by using the *predict* method for each subject we are able to get the output that indicates in which cluster the subject inserts itself. Therefore, with the hashmap and this output we can access the subject and the folders. By creating a copy of the initial email file, we are able to introduce the email in the correct folder through the use of the *cp* command.

At the end of this process, the folders with the clusters will contain the text files with the emails that the clustering model classified. Figure 3.4 includes a scheme that aims to explain the process referred above in a more friendly way.

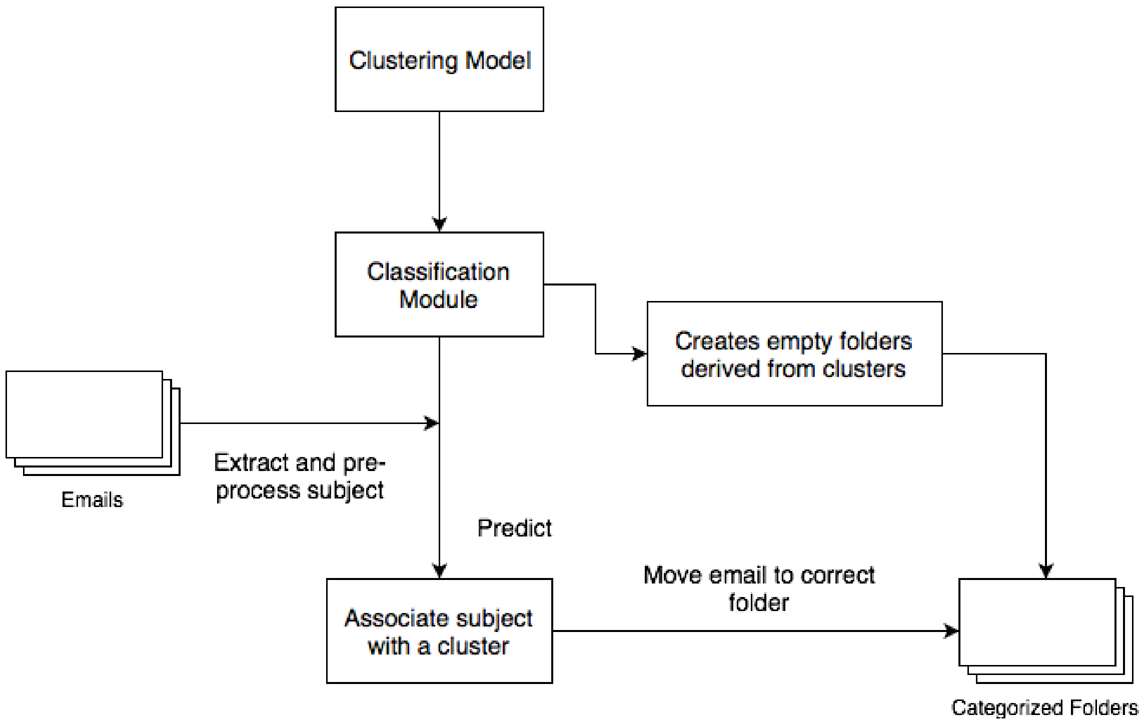
Clustering Model
Classification Module
Creates empty folders derived from clusters
Emails
Extract and pre-process subject
Predict
Move email to correct folder
Associate subject with a cluster
Categorized Folders

Figure 3.4: Email classification module

Chapter 4

Experiments and Results

The main goal of this chapter is to document the developed experiments and to fetch conclusions about the obtained results in order to achieve one of this project's goals: a comparison between different clustering algorithms applied to emails present in AMT-Consulting inboxes.

This dataset includes six inboxes with different sizes and with a different number of emails. The general data regarding this dataset may be seen in the Table 4.1.

In the following sections, several experiments will be conducted in order to obtain conclusions about the results. The elements included in the experiments are the metrics used to measure the clustering model quality such as *silhouette* score and the algorithm's running time, the feature extraction methods used in this process that include *bag of words*, *tf-idf*, *word2vec* and *doc2vec* and the clustering algorithms used that comprise K-means, DBSCAN and agglomerative clustering, in other words, one algorithm for each main clustering category: partitional, density-based and hierarchical clustering.

Inbox	**Data Size (Mb)**	**Number of emails**
Inbox1	6164.48	12585
Inbox2	597.6	1570
Inbox3	36	45
Inbox4	261.1	894
Inbox5	49	96
Inbox6	916.8	8065

Table 4.1: Email inboxes datas

4.1 Experimental Setup

Before the experiments execution, the inboxes were previously treated in order conduct the experiments in an efficient way.

Firstly, all the six *.pst* files were stored in a folder called "inboxes". Then, for each inbox, the *libpst utilities* library was used to convert the *.pst* file to several *.mbox* files where each *.mbox* corresponds to one folder that the inbox contained. This files were stored in a folder called "email_mboxes". However, not every folder was available for extraction, folders such as "Sent Items", "Drafts" and "Deleted" were excluded from the extraction process as well as the files with ".calendar" extension.

Secondly, for each ".mbox" file, the emails were extracted and two import folders were created. One first folder called "emails" that contains the emails content, in other words, this folder contains text files in which the file name correponds to the email's subject concatenated with its date. The second folder is the most important one for the experimental process since it contains the data that will be used in the clustering process. This folder is called "processed_emails" and it includes one text file for each inbox. This text file contains all the email subjects extracted from all the emails related with a specific inbox.

The final setup process directory tree would look like the scheme presented in Figure 4.1.

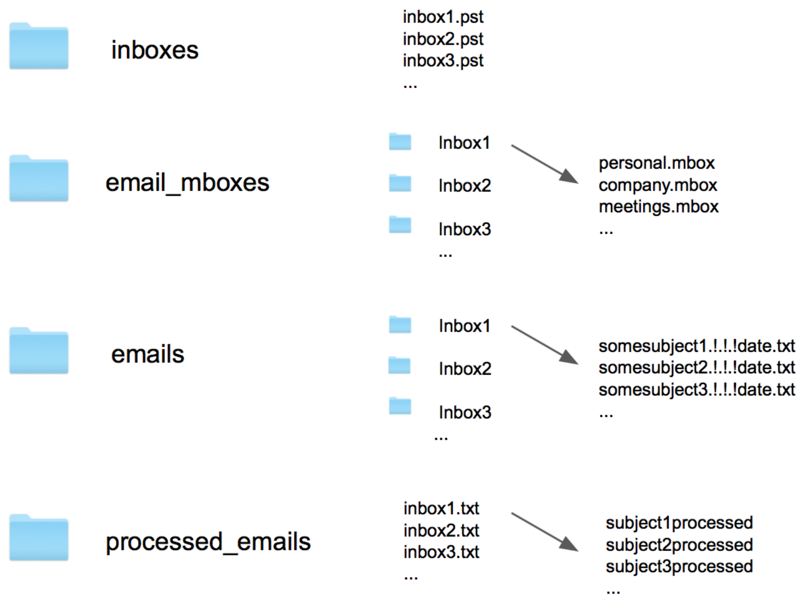

Figure 4.1: Experimental setup folders

4.2 First scenario - K-means

In this first scenario, it was used the K-means algorithm. The implementation was available in *scikit-learn* library and the version that was executed was K-means++. For now, the algorithms run sixteen times and the number of clusters was changed in each execution. These executions started with the number of clusters set to four and the program run through a loop in which each execution incremented the number of clusters until this number was equal to twenty. The referred limits (four and twenty) are meant to reveal a realistic number of folders that a user would like to use since three email folders is a very small number and more than twenty folders constitute an high number of folders to organize the emails into.

The referred approach was conducted for each one of the six inboxes which permits the analysis of this algorithm in datasets with different sizes.

It is also important to refer that this execution was not only made for each inbox but also for each feature selection method including, *bag of words*, *tf-idf*, *word2vec* and *doc2vec*.

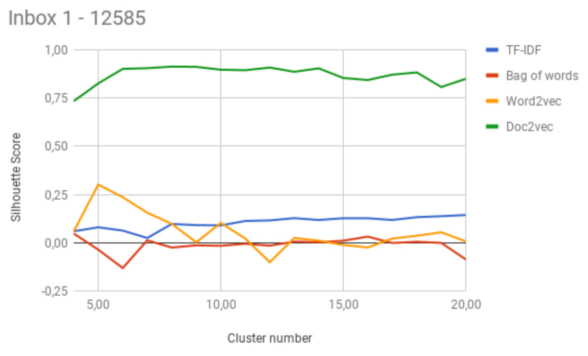

Figure 4.2: First scenario - Inbox1

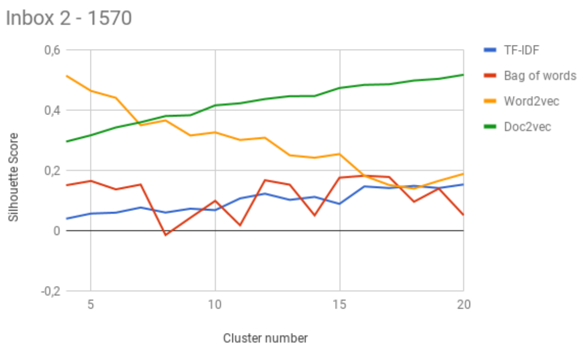

Figure 4.3: First scenario - Inbox2

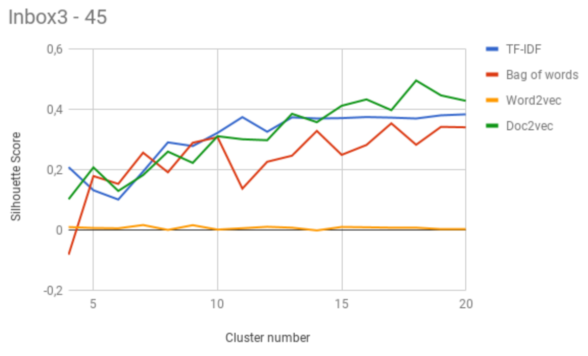

Figure 4.4: First scenario - Inbox3

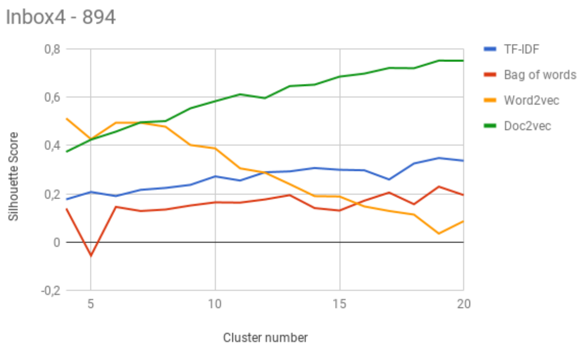

Figure 4.5: First scenario - Inbox4

The charts present in Figures 4.2, 4.3, 4.4, 4.5, 4.6 and 4.7 include the *silhouette* score results obtained using K-means with different feature extraction methods and each chart is associated with one inbox (the name of the inbox and the number of emails it contains are explicit in the chart's title).

Through the analysis of these results, we are able to conclude the following for each feature extraction method:

- **Bag of words**: bag of words is feature extraction method with worst results because it has the lowest *silhouette* scores in several inboxes with the exception of inbox 3 and 5 which are the ones with fewer emails which makes sense that the word2vec and doc2vec measures have fewer score than bag of words measure.

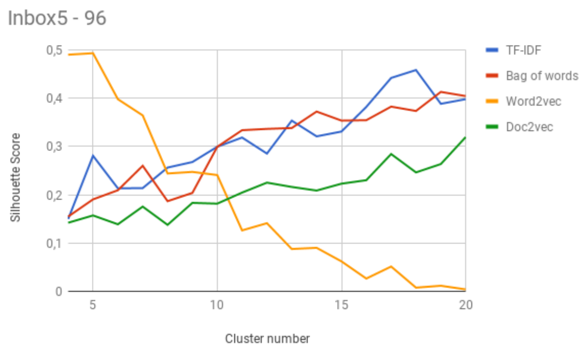

Figure 4.6: First scenario - Inbox5

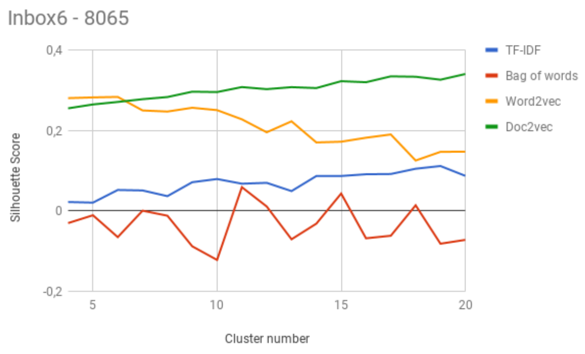

Figure 4.7: First scenario - Inbox6

- **Tf-idf**: *tf-idf* is a much more stable feature extraction method since its scores usually maintained a consistent result among the different inboxes with different sizes with significantly better than results than *bag of words* score.

- **Word2vec**: *word2vec* is very unstable. One of the possible reasons behind this instability may be the extreme sensitivity to the change of the size of the dataset despite the fact that the algorithm's parameters change accordingly to the inbox's size count. One other reason that may cause this is the few training data used in this neural network due to the low number of emails inside the inoxes.

- **Doc2vec**: the *doc2vec* presents itself as a good option for inboxes with an high number of emails and the reason may be the fact that it has much more material to train the neural network which will create vectors with more quality and consequently better clustering results.

4.3 Second scenario - Agglomerative clustering

As it was referred in Section 2.4.3, agglomerative clustering has a very different approach when compared to K-means algorithm, therefore, this scenario will present several differences when compared to the first one.

In this scenario, it was used only the *tf-idf* method and new variables were introduced in the comparisons. Agglomerative clustering works by successively merging clusters and there are several ways to calculate distances between clusters in order to merge them. Cluster distance measure techniques will be compared in this scenario which include *ward*, *complete* and *average*. One other important part of agglomerative clustering execution is the distance measure used in the clustering distance technique which will also be compared by executing the algorithm with *cosine* and *euclidean* distances. However, the *cosine* distance is not available for the *ward* implementation but only for *complete* and *average*.

The agglomerative clustering implementation contains a parameter called "number of clusters", however, this number has a different meaning from the one in K-means. In agglomerative clustering, the "number of clusters" parameter represents the number of clusters that are meant to be merged, in other words, the algorithm starts by merging the clusters and when it achieves a certain number, it stops its execution. In this scenario several runs were made with different numbers of clusters from 4 to 20.

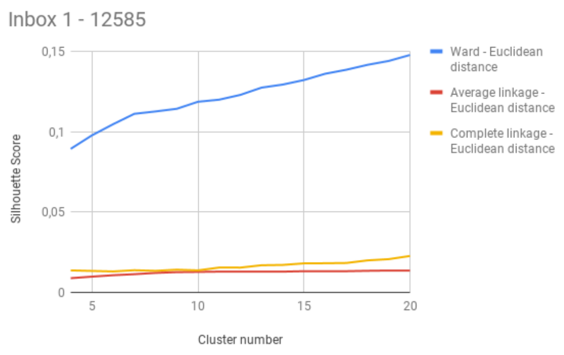

Figure 4.8: Second scenario - Inbox1

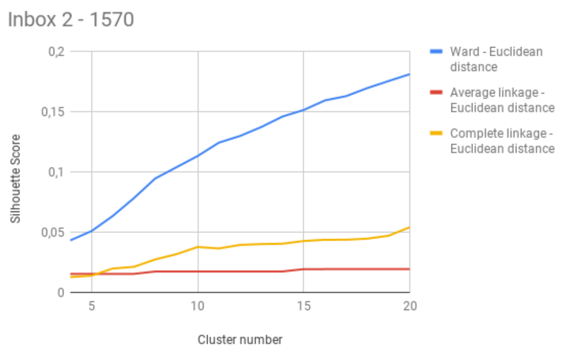

Figure 4.9: Second scenario - Inbox2

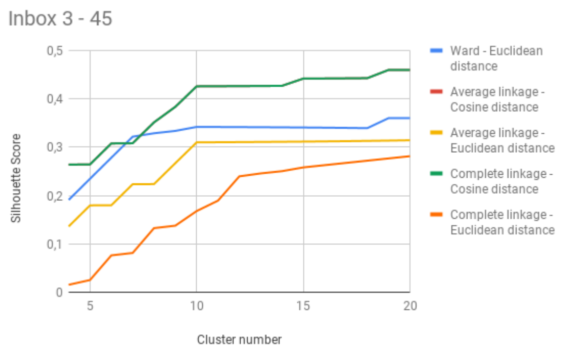

Figure 4.10: Second scenario - Inbox3

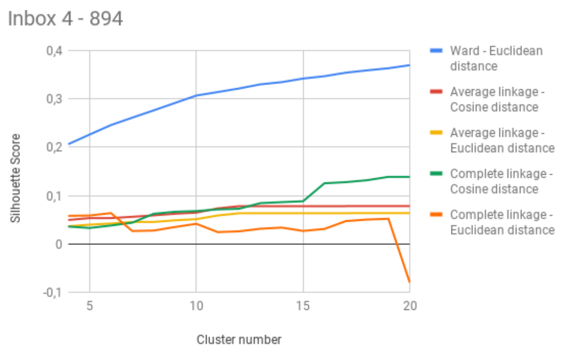

Figure 4.11: Second scenario - Inbox4

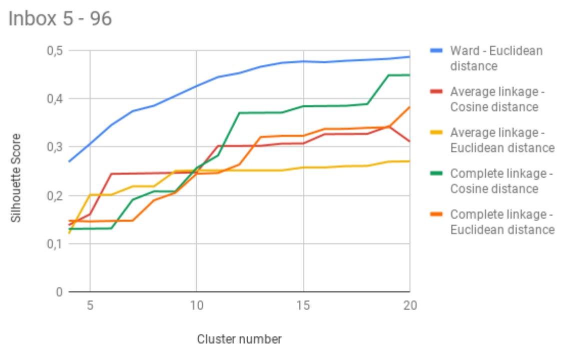

Figure 4.12: Second scenario - Inbox5

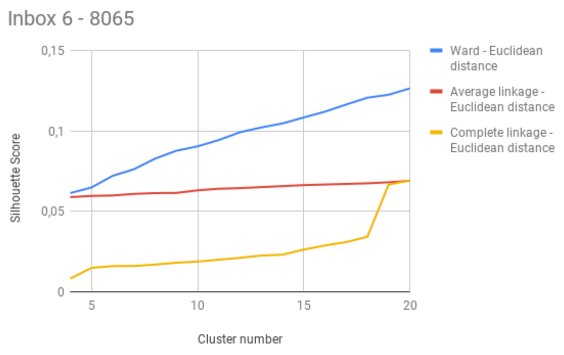

Figure 4.13: Second scenario - Inbox6

With the results obtained in Figures 4.8, 4.9, 4.10, 4.11, 4.12 and 4.13, we are able to compare the three merging clustering methods and the best distances measures to use. However, it is also important to say that the cosine distance was not viable to apply in inbox 1, 2 and 6 due to implementation issues regarding the used library.

Firstly, the most important conclusion to be retrieve from the charts is related with the "ward" approach. In all the inboxes, except in inbox 3, it obtained the best *silhouette* score, especially in inbox 1, 2 and 4. With this observation we can say that the "ward" merging technique is the best option to use in the agglomerative clustering approach.

Regarding the other merging types, the results are very dependent on the data size and text content because we have several variances in the data where the *average* linkage produces a better *silhouette* score and other situations where it produces worse scores than the *complete* linkage. For example, in inbox 6, there is a better score for *average* linkage compared with *complete* linkage on the first 18 executions. However, in inbox 1, the scores of these two approaches are very similar.

4.4 Third scenario - DBSCAN

DBSCAN is also a very different approach from the algorithms implemented above. One of those differences is that the number of clusters is not an input but instead it is a number that depends on the density of the dataset. This way, the created scenario was meant to vary DBSCAN arguments in order to obtain different numbers of clusters and evaluate clustering model quality. The feature selection technique used in these clustering executions was *tf-idf*.

Parameters		**Inbox 1**		**Inbox 2**		**Inbox 3**	
N	**min**	**Nr. clusters**	**Silhouette**	**Nr. clusters**	**Silhouette**	**Nr. clusters**	**Silhouette**
0.6	**2**	1643	0.714210	212	0.412782	8	0.277615
0.9	**2**	1164	0.433228	168	0.2744748	7	0.228644
0.9	**5**	410	0.312371	51	0.235933	2	-0.053253
1	**2**	93	-0.017654	16	0.000437	2	0.092271
0.9	**20**	73	0.169007	10	0.118539	-	-
0.9	**30**	48	0.159080	8	0.103683	-	-
0.9	**50**	20	0.129174	1	0.023702	-	-
0.9	**60**	16	0.123267	1	0.026452	-	-
0.9	**80**	10	0.115828	-	-	-	-

Table 4.2: Results DBSCAN: 1,2,3.

One of the first conclusions that we may take from Table 4.2 and Table 4.3, is that if we reduce the *min_sample*, the number of clusters will increase which makes sense because if minimum number of samples is lower, the will be easier to create a dense zone and, consequently, a cluster.

Secondly, it is common, specially for the inboxes with an higher amount of emails to have a better *silhouette* score, however, several inboxes with a lower email count, will not be able to run with the specified arguments due to the nature of the dataset, nevertheless, they will provide a term of comparison with the K-means and agglomerative clustering approach if we set these algorithms to have the same number of clusters as the ones that DBSCAN created.

Parameters		**Inbox 4**		**Inbox 5**		**Inbox 6**	
N	**min**	**Nr. clusters**	**Silhouette**	**Nr. clusters**	**Silhouette**	**Nr. clusters**	**Silhouette**
0.6	**2**	1643	0,553232	14	0,504079	1189	0,797418
0.9	**2**	1164	0,460328	13	0,475583	1019	0,628827
0.9	**5**	410	0,258393	5	0,345391	357	0,336556
1	**2**	93	0,057733	13	0,476286	172	0,059561
0.9	**20**	73	0,245898	-	-	46	-0,021028
0.9	**30**	48	0,177360	-	-	24	-0,084202
0.9	**50**	20	0,149563	-	-	15	-0,131029
0.9	**60**	16	0,149563	-	-	10	-0,154036
0.9	**80**	10	0,149563	-	-	5	-0,190674

Table 4.3: Results DBSCAN: 4,5,6.

4.5 Execution time comparison

In this section the execution time comparison will be done in an iterative way. Firstly, we compare the running times between K-means and agglomerative clustering, then, we compare DBSCAN with agglomerative and K-means making it a triple comparison. This comparison should be done this way because DBSCAN has a few different characteristics that should be taken into account on the comparison part such as the fact that the number of clusters does not constitute a parameter. It is also important to refer that the linkage method for agglomerative clustering used in this scenario will be *ward* since it was the one that obtain the best results in Section 4.3.

Therefore, we will compare each one of these scenarios for the six different inboxes.

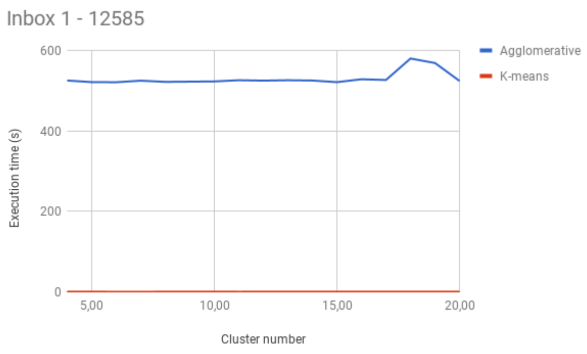

Figure 4.14: Execution time - Inbox 1

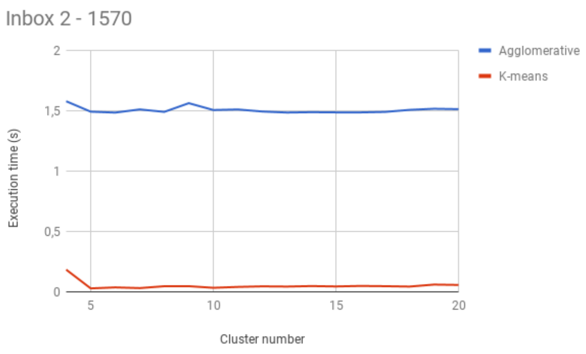

Figure 4.15: Execution time - Inbox 2

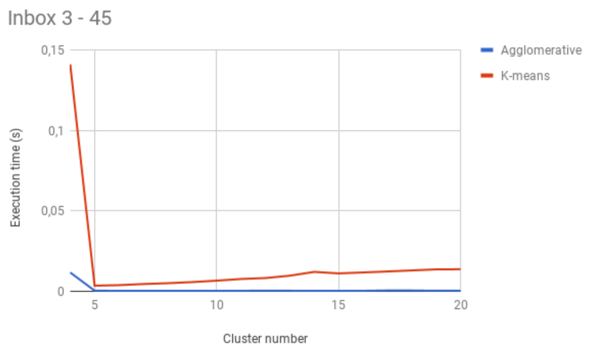

Figure 4.16: Execution time - Inbox 3

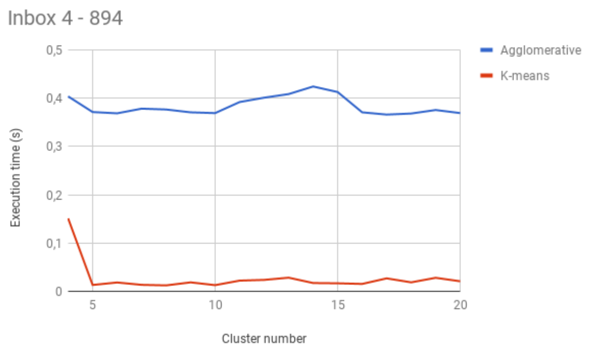

Figure 4.17: Execution time - Inbox 4

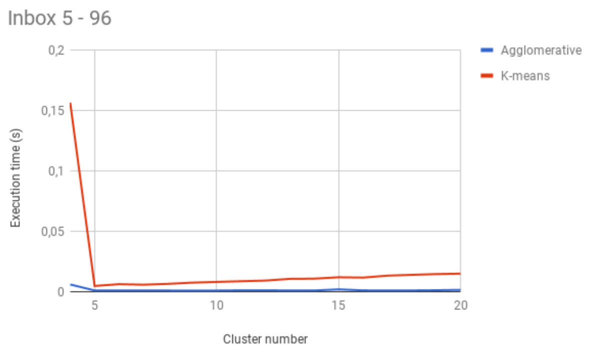

Figure 4.18: Execution time - Inbox 5

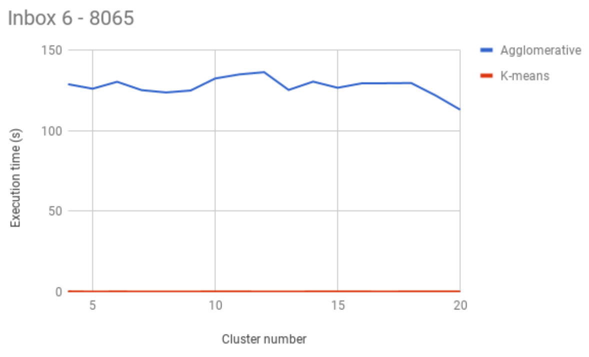

Figure 4.19: Execution time - Inbox 6

From the results presented in Figures 4.14, 4.15, 4.16, 4.17, 4.18 and 4.19, we can conclude that the K-means algorithm has higher running times in email boxes with a small amount of emails, however, for inboxes with an higher amount of emails like inbox 1, 2, 4 and 6, the agglomerative clustering running time increases exponentially due to its computational demands.

In regard to the DBSCAN algorithm, we should take a different approach. In this comparison we will take the number of clusters that the algorithm created and comparing the running times with the number of clusters also used in K-means and agglomerative clustering.

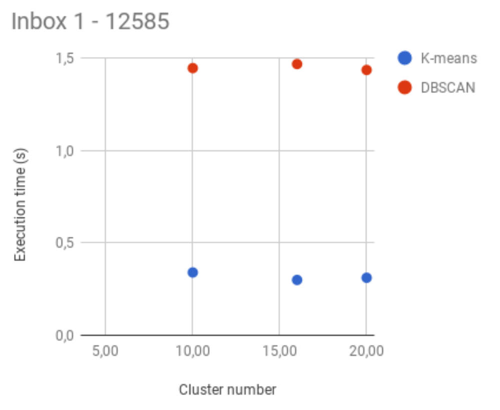

Figure 4.20: Execution time w/DBSCAN - Inbox 1

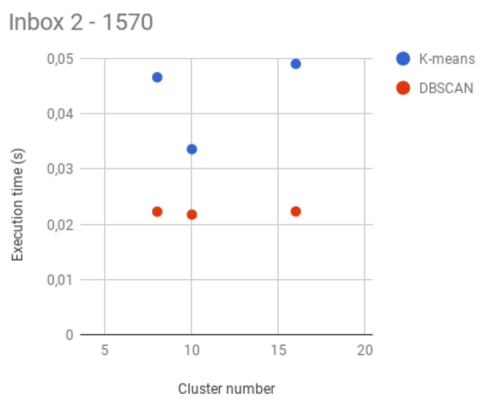

Figure 4.21: Execution time w/DBSCAN - Inbox 2

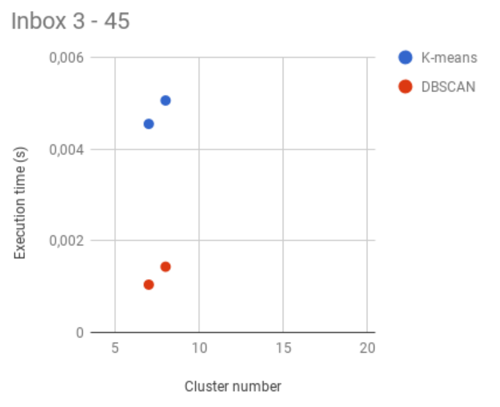

Figure 4.22: Execution time w/DBSCAN - Inbox 3

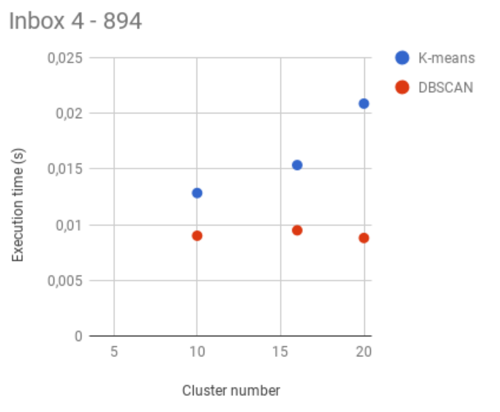

Figure 4.23: Execution time w/DBSCAN - Inbox 4

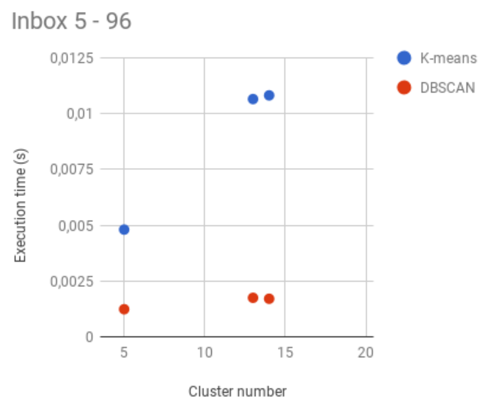

Figure 4.24: Execution time w/DBSCAN - Inbox 5

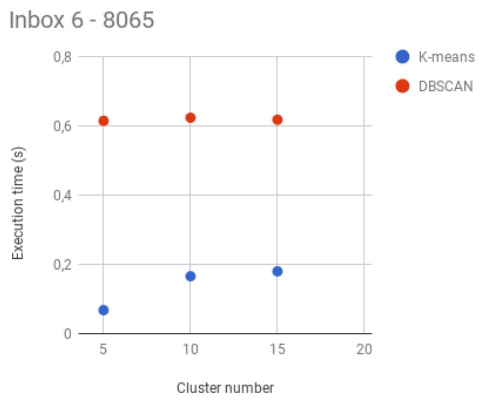

Figure 4.25: Execution time w/DBSCAN - Inbox 6

From the retrieved data explicit in Figures 4.20, 4.21, 4.22, 4.23, 4.24, and 4.25, we can infer that K-means has a higher running time for smaller datasets while DBSCAN has an higher running time in inboxes with an higher number of emails. Therefore, K-means algorithm has better performance than DBSCAN for large datasets while DBSCAN has better performance in smaller ones.

4.6 *Silhouette* score comparison between algorithms

As it was mentioned in Section 2.4.4, the *silhouette* score is one of the internal clustering evaluation measures that aims to evaluate the clusters compactness and separation. Therefore, it is important to retrieve this metric from the created clustering models in order to make a comparison between the used types of clustering algorithms.

The approach used to compare the different algorithms *silhouette* scores is similar to one used to measure running times. Firstly, we compare K-means with agglomerative clustering which can be compared in an easier way and, then, we compare the DBSCAN executions that contain the same number of clusters as the other two algorithms.

The following charts present a comparison of the *silhouette* scores retrieved from the executions of all the six email inboxes between agglomerative clustering using *ward* as the *linkage* method and *euclidean* distance as the affinity approach and K-means. Both algorithms used *tf-idf* as the feature extraction method.

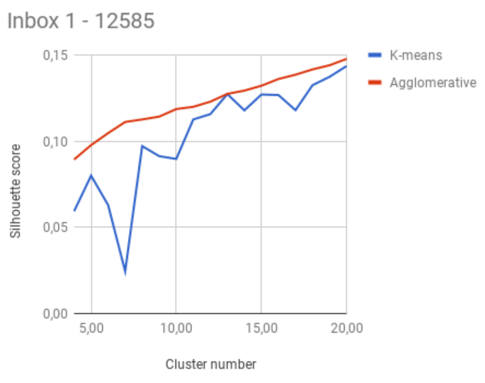

Figure 4.26: Silhouette score - Inbox 1

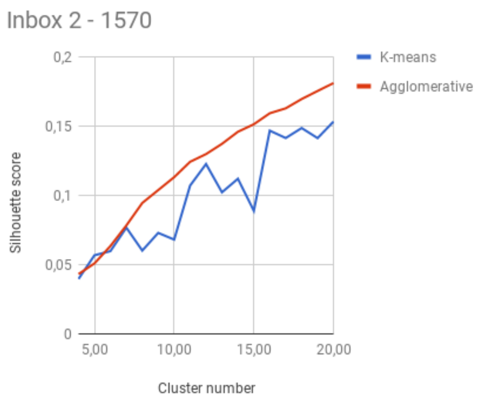

Figure 4.27: Silhouette score - Inbox 2

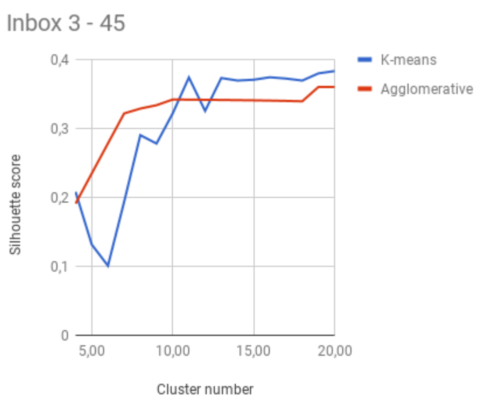

Figure 4.28: Silhouette score - Inbox 3

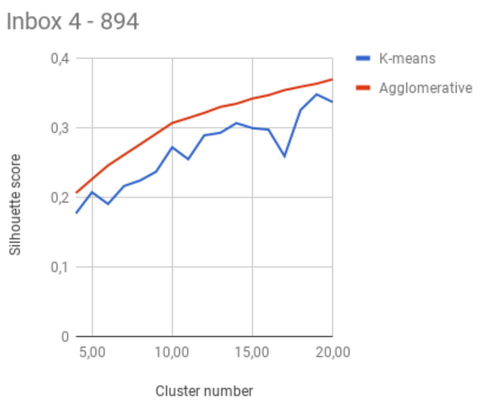

Figure 4.29: Silhouette score - Inbox 4

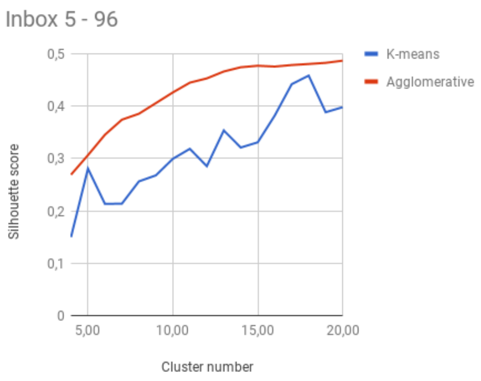

Figure 4.30: Silhouette score - Inbox 5

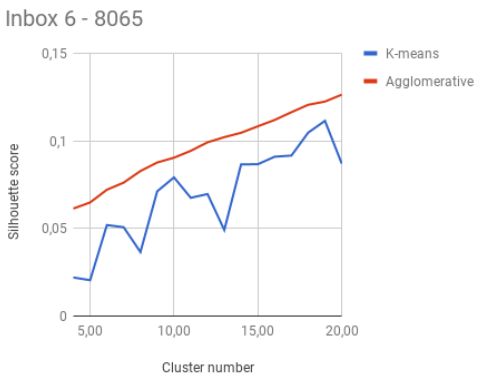

Figure 4.31: Silhouette score - Inbox 6

As we can see in the charts present in Figures 4.26, 4.27, 4.28, 4.29, 4.30 and 4.31, the agglomerative clustering algorithm has an higher *silhouette* score along the different number of clusters and throughout the different inboxes with an exception in the inbox 3 where the K-means algorithm surpasses the agglomerative clustering score in the model with 12 clusters and beyond. This means that the models created by the agglomerative clustering algorithm are more compact and separated.

The agglomerative clustering algorithm is also more consistent since it maintains a better evolution along the number of clusters while K-means suffers from several drops in the score due to its variation in the initial positions of the centroids.

However, it is also important to make a comparison with the algorithm DBSCAN. As it was mentioned before, the same approach that was taken in the measurement of the execution times will also be applied in this *silhouette* score analysis. The results related with the *silhouette* score comparing the three algorithms are explicit in Figures 4.32, 4.33, 4.34, 4.35, 4.36 and 4.37.

Despite the fact that there are few instances of the model between the three algorithms, we can say that agglomerative clustering is still the best competitor because it presents the best scores in inbox 1, 3 and 4. However, we can say the DBSCAN is a very unpredictable algorithm because,

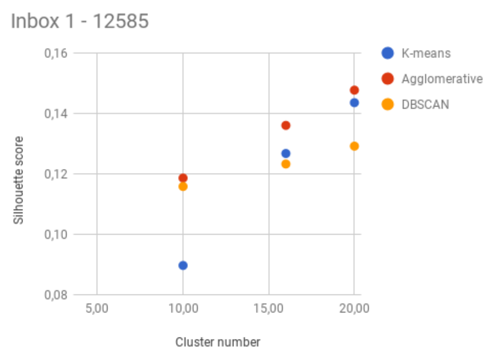

Figure 4.32: Silhouette score w/DBSCAN - Inbox 1

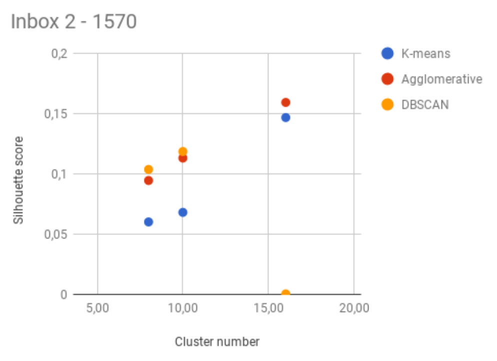

Figure 4.33: Silhouette score w/DBSCAN - Inbox 2

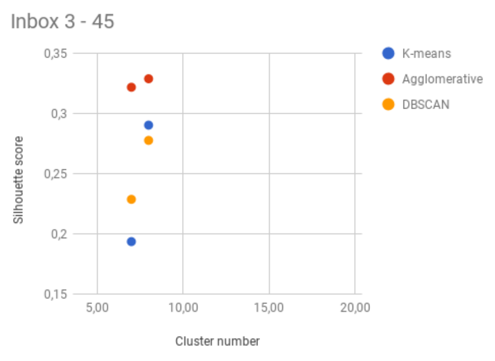

Figure 4.34: Silhouette score w/DBSCAN - Inbox 3

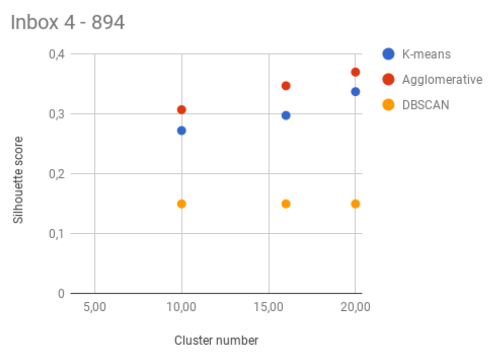

Figure 4.35: Silhouette score w/DBSCAN - Inbox 4

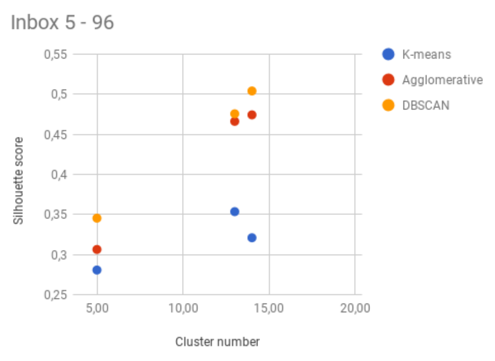

Figure 4.36: Silhouette score w/DBSCAN - Inbox 5

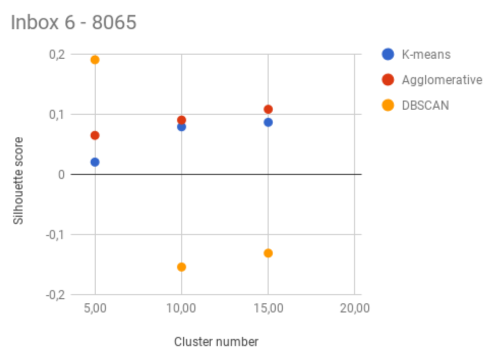

Figure 4.37: Silhouette score w/DBSCAN - Inbox 6

for example, in inbox 6, it can have the highest *silhouette* score when the number of clusters is 5 but it also has the lowest and worse *silhouette* scores of all models the were executed as we can see in the cluster number 10 and 15.

4.7 *Silhouette* score comparison between inboxes

In order to obtain a better understanding of the six different datasets and their clustering behaviour, an higher number of clusters was used to created models. These models used *tf-idf* as feature extraction method and K-means as the clustering algorithm. The *silhouette* score was measured for each run and the results may be seen in Figures 4.38 and 4.39.

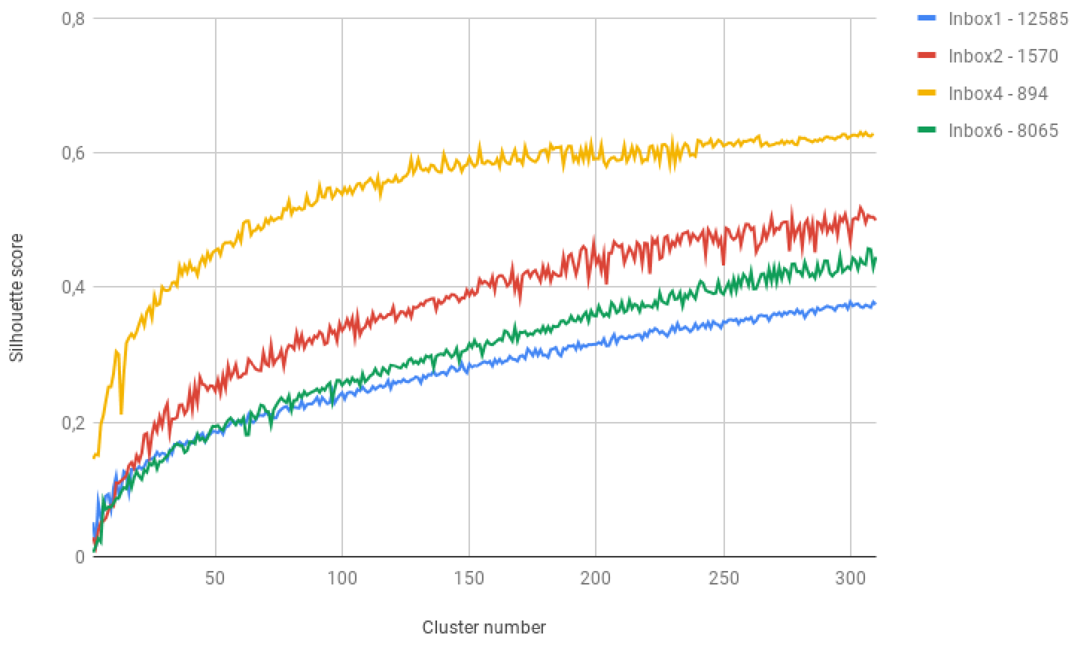

Figure 4.38: Silhouette score comparison between inboxes 1, 2, 4 and 6

This comparison includes two charts separated by its size. While in Figure 4.38 we can find the *silhouette* score variation for the inboxes with a higher amount of email data (inboxes 1, 2, 4 and 6), Figure 4.39 contains the inboxes with fewer emails (inboxes 3 and 5).

With the information present in these charts, we can infer that the variation between the number of clusters and the *silhouette* score follows a similar format a logarithmic function, which means that the *silhouette* score suffers an initial accentuated growth and, then, it stabilizes.

Two main conclusions may be retrieved from these charts.

Firstly, the higher amount of emails that inbox contains, the lower the *silhouette* score will be for the latter hundreds of clusters (Figure 4.38). This is mainly due to the fact that we are trying to cluster the dataset with a small amount of clusters compared to number of emails contained in the inbox, therefore, the higher the number of clusters compared with the size of the inbox, the higher is its *silhouette* score, which leads to better compactness and separation in the models.

Secondly, we are also able to conclude that the use of K-means algorithm leads a an unstable growth throughout the number of clusters with small peaks and lows. One of the causes may be the different initial centroid positions of the model. This problem may be solved by running K-means

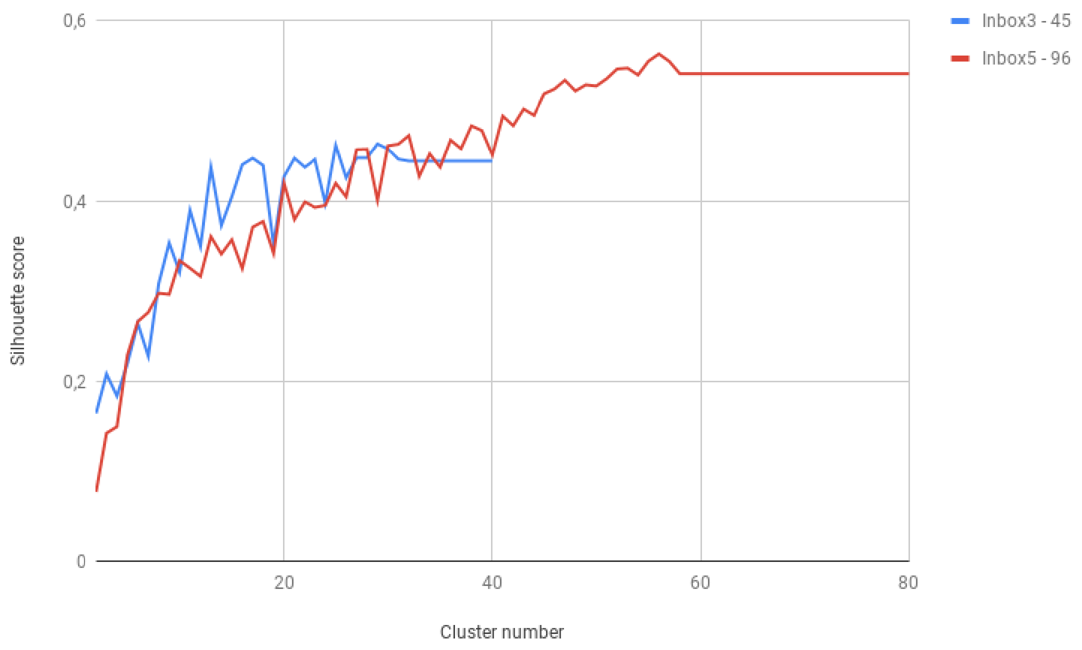

Figure 4.39: Silhouette score comparison between inbox 3 and 5

several times until it reaches the higher score, which means that the initial centroids positions were optimal and the model should be saved.

4.8 Discussion

From the first scenario we can conclude that *tf-idf* the metric is the best feature extraction method to be used in this specific context, because it presents more stable results among data with different characteristics. When compared to the *bag of words* approach, *tf-idf* achieved a higher *silhouette* score for most inboxes. Due to its instability, *word2vec* was excluded from the viable options to be used in the categorization system. *Doc2vec* proved to be a good option for inboxes with a higher amount of email data, however, for inboxes with a lower email size, it suffered a drop in the *silhouette* score, which was lower than *bag of words* and *tf-idf*.

Therefore, concerning the feature extraction method, we should opt for *tf-idf* because it proved to be the more stable approach. Since this system categorizes inboxes with different sizes, we choose the method that presents more stable results, rather than the one with the best results only for inboxes with an high amount of email data.

Regarding the three used clustering algorithms, we can say that each one of them has its own advantages and disadvantages.

From the obtained results, we saw that when comparing K-means with agglomerative clustering, the latter achieved better *silhouette* score results, however, the execution time grew accordingly with the inbox size, while K-means executions times remained very low. Nonetheless, the *silhouette* score difference between these two algorithms may not compensate the increase in the running time, which leads to the choice of K-means over agglomerative clustering.

When comparing DBSCAN with K-means, it is important to refer that a choice between these two algorithms lies beyond the obtained results due to the different characteristics of the algorithms. Despite the fact that the comparing data regarding the DBSCAN algorithm was small, it shown negative results in inbox 6 regarding the *silhouette* score and irregular results in the other inboxes.

Although the results point to K-means as the best option, there is another important aspect to take into account, being this the parameters that each one of the algorithms receive. If the user wants to specify the number of folders that he wants to organize the inbox with, then, he should opt for K-means. Although DBSCAN does not need the number of clusters as input, several parameters should be inserted in order to run the algorithm (*N* and *min_samples*), and since DBSCAN is very sensible to these parameters, K-means presents itself as the best option to be used in the email classification system.

Chapter 5

Conclusions and Future Work

In conclusion, we can infer that the main objectives of this dissertation were accomplished: the development of an email categorization system prototype and the comparison between unsupervised learning techniques.

The whole process of email categorization was conceived including the specific tasks necessary to integrate it with the data format that was given (*pst* files). Therefore, from the extraction of email content, up until the topic extraction and passing through data preparation, feature extraction and clustering tasks, the system is able to receive a *pst* file and automatically create folders containing the emails separated by topics.

Regarding the comparison between unsupervised learning techniques, it is possible to say that the obtained conclusions with the results analysis may be useful for future projects regarding text categorization techniques. *Tf-idf* is a good option for the feature extraction process in clustering contexts where one of the goals is topic extraction. *Doc2vec* is also a promising feature extraction method but it has the training limitation that requires large amounts of data in order to obtain good results.

We also concluded that K-means has the highest proximity between performance and cluster quality, and that agglomerative clustering has slightly better results but its very computational demanding which represents a disadvantage.

It is also important to refer that these conclusions are not related with the general view of text clustering but rather with a very specific context regarding the datasets from AMT-Consulting collaborators.

Lastly, we can say that this dissertation project and conclusions may be useful for future unsupervised classification projects that could be developed in AMT-Consulting.

5.1 Future work

Despite the fact that the main objectives of this work were accomplished, there were several limitations along the development related with the dataset.

Initially, this project should have a component regarding supervised learning techniques. However, the nature of the data did not permit such research due to the lack of sufficient data for neural network training, which led to adjustments in the project's goals. Therefore, if, in the future, labeled datasets are available in AMT-Consulting context, it would be interesting to make a comparison between the results obtained using unsupervised learning techniques with supervised learning methods.

On the other hand, several features could be added to the created prototype, such as an user interface with the possibility for the user to choose the preferred algorithm or how many categories he would like the system to generate.

Regarding the comparison between algorithms, it may also be taken to a next level by comparing other types of clustering algorithms such us fuzzy clustering.

References

[AA15] Izzat Alsmadi and Ikdam Alhami. Clustering and classification of email contents. *Journal of King Saud University - Computer and Information Sciences*, 27(1):46–57, 2015.

[AAU$^+$17] S. Akter, A. S. Asa, M. P. Uddin, M. D. Hossain, S. K. Roy, and M. I. Afjal. An extractive text summarization technique for bengali document(s) using k-means clustering algorithm. In *2017 IEEE International Conference on Imaging, Vision Pattern Recognition (icIVPR)*, pages 1–6, Feb 2017.

[Aiz03] Akiko Aizawa. An information-theoretic perspective of tf–idf measures. *Information Processing & Management*, 39(1):45–65, jan 2003.

[ASFA17] Sigit Adinugroho, Yuita Arum Sari, M. Ali Fauzi, and Putra Pandu Adikara. Optimizing K-means text document clustering using latent semantic indexing and pillar algorithm. *2017 5th International Symposium on Computational and Business Intelligence (ISCBI)*, pages 81–85, 2017.

[AV07] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 1027–1025, 2007.

[BMH04] Ron Bekkerman, Andrew McCallum, and Gary Huang. Automatic Categorization of Email into Folders : Benchmark Experiments on Enron and SRI Corpora. *Science*, 418:1–23, 2004.

[CYTZ09] Xiuguo Chen, Wensheng Yin, Pinghui Tu, and Hengxi Zhang. Weighted k-Means Algorithm Based Text Clustering. *2009 International Symposium on Information Engineering and Electronic Commerce*, pages 51–55, 2009.

[EKSX96] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters a density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pages 226–231. AAAI Press, 1996.

[GBR15] Markus Götz, Christian Bodenstein, and Morris Riedel. HPDBSCAN: highly parallel DBSCAN. *Proceedings of the Workshop on Machine Learning in High-Performance Computing Environments*, page 2, 2015.

[Gen] Gensim. Api reference. https://radimrehurek.com/gensim/apiref.html. Accessed: 2018-03-16.

[Gha04] Zoubin Ghahramani. Unsupervised learning. In *Advanced Lectures on Machine Learning*, pages 72–112. Springer-Verlag, 2004.

[GSGR17] Harsha S. Gowda, Mahamad Suhil, D. S. Guru, and Lavanya Narayana Raju. Semi-supervised text categorization using recursive k-means clustering. *CoRR*, abs/1706.07913, 2017.

[IPP16] Jasmine Irani, Nitin Pise, and Madhura Phatak. Clustering Techniques and the Similarity Measures used in Clustering: A Survey. *International Journal of Computer Applications*, 134(7):975–8887, 2016.

[iW03] Sabine Schulte im Walde. Chapter 4 Clustering Algorithms and Evaluations. *Clustering Algorithms and Evaluations*, (1973):51–69, 2003.

[JD88] Anil K. Jain and Richard C. Dubes. *Algorithms for clustering data*. Prentice Hall, 1988.

[JMF99] A. K. Jain, M. N. Murty, and P. J. Flynn. Data clustering: A review. *ACM Comput. Surv.*, 31(3):264–323, September 1999.

[KB14] Gaurav Kumar and Pradeep Kumar Bhatia. A detailed review of feature extraction in image processing systems. In *Proceedings of the 2014 Fourth International Conference on Advanced Computing & Communication Technologies*, ACCT '14, pages 5–12, Washington, DC, USA, 2014. IEEE Computer Society.

[KKR15] Ch. Bala Koteshwariah, N. Raghu Kisore, and V. Ravi. A fuzzy version of generalized DBSCAN clustering algorithm. *Proceedings of the Second ACM IKDD Conference on Data Sciences - CoDS '15*, pages 128–129, 2015.

[KRA$^+$14] K. Khan, S.U. Rehman, Kamran Aziz, Simon Fong, and S. Sarasvady. DBSCAN : Past , Present and Future. *Applications of Digital Information and Web Technologies (ICADIWT), 2014 Fifth International Conference on*, pages 232–238, 2014.

[Kwa08] Nojun Kwak. Feature extraction for classification problems and its application to face recognition. *Pattern Recognition*, 41(5):1718–1734, may 2008.

[KY04] Bryan Klimt and Yiming Yang. The Enron Corpus: A New Dataset for Email Classification Research. pages 217–226. Springer, Berlin, Heidelberg, 2004.

[Lew92] David D. Lewis. Feature selection and feature extraction for text categorization. In *Proceedings of the Workshop on Speech and Natural Language*, HLT '91, pages 212–217, Stroudsburg, PA, USA, 1992. Association for Computational Linguistics.

[LL16] Hui Li and Qing Li. Forum topic detection based on hierarchical clustering. *2016 International Conference on Audio, Language and Image Processing (ICALIP)*, pages 529–533, 2016.

[LM14] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, pages II–1188–II–1196. JMLR.org, 2014.

[LY07] Chung Hong Lee and Hsin Chang Yang. Implementation of unsupervised and supervised learning systems for multilingual text categorization. *Proceedings - International Conference on Information Technology-New Generations, ITNG 2007*, pages 377–382, 2007.

[MBMM17] Andrea Morichetta, Enrico Bocchi, Hassan Metwalley, and Marco Mellia. CLUE: Clustering for mining web URLs. *Proceedings of the 28th International Teletraffic Congress, ITC 2016*, 1:286–294, 2017.

[MC11] Fionn Murtagh and Pedro Contreras. Methods of hierarchical clustering. *CoRR*, abs/1105.0121, 2011.

[MCCD13] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv preprint arXiv:1301.3781*, pages 1–12, 2013.

[Mic] Microsoft. Introduction to outlook data files (.pst and .ost). https://support.office.com/en-us/article/introduction-to-outlook-data-files-pst-and-ost-222eaf92-a995-45d9-bde2-f331f60e2790. Accessed: 2018-05-28.

[MSC$^+$13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed Representations of Words and Phrases and Their Compositionality. In *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'13, pages 3111–3119, USA, 2013. Curran Associates Inc.

[MZ15] Long Ma and Yanqing Zhang. Using Word2Vec to Process Big Text Data. In *Proceedings of the 2015 IEEE International Conference on Big Data (Big Data)*, BIG DATA '15, pages 2895–2897, Washington, DC, USA, 2015. IEEE Computer Society.

[PD15] Deepa Patil and Yashwant Dongre. A Clustering Technique for Email Content Mining. *International Journal of Computer Science and Information Technology*, 7(3):73–79, 2015.

[PVG$^+$11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[RK17] Raihannur Reztaputra and Masayu Leylia Khodra. Sentence structure-based summarization for Indonesian news articles. *Proceedings - 2017 International Conference on Advanced Informatics: Concepts, Theory and Applications, ICAICTA 2017*, pages 0–5, 2017.

[Rou87] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20(C):53–65, 1987.

[RU11] Anand Rajaraman and Jeffrey David Ullman. Data Mining. *Mining of Massive Datasets*, 18 Suppl:7–15, 2011.

[Sai15] D Sailaja. An Overview of Pre-Processing Text Clustering. *International Journal of Computer Science and Information Technologies,*, 6(3):3119–3124, 2015.

[SAJ17] Ma. Shiela C. Sapul, Than Htike Aung, and Rachsuda Jiamthapthaksin. Trending topic discovery of Twitter Tweets using clustering and topic modeling algorithms. In *2017 14th International Joint Conference on Computer Science and Software Engineering (JCSSE)*, pages 1–6. IEEE, jul 2017.

[SB88] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Inf. Process. Manage.*, 24(5):513–523, August 1988.

[SB13] K Sasirekha and P Baby. Agglomerative Hierarchical Clustering Algorithm-A Review. *International Journal of Scientific and Research Publications*, 3(1):2250–3153, 2013.

[Sha09] Cosma Shalizi. Distances between Clustering , Hierarchical Clustering. *Data Mining*, (September):36–350, 2009.

[Sie15] Scharolta Katharina Sien. Adapting word2vec to Named Entity Recognition. *Proceedings of the 20th Nordic Conference of Computational Linguistics (NODALIDA 2015)*, (Nodalida):239–243, 2015.

[SK17] Krithi Shetty and Jagadish S. Kallimani. Automatic extractive text summarization using K-means clustering. *2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)*, pages 1–9, 2017.

[Sl13] Scikit-learn. 4.2. feature extraction. `http://scikit-learn.org/stable/modules/feature_extraction.html`, 2013. Accessed: 2018-05-15.

[Smi] David Smith. libpst utilities - version 0.6.71. http://www.five-ten-sg.com/libpst/rn01re01.html. Accessed: 2018-03-01.

[SSAV17] Aakanksha Sharaff, Hari Shrawgi, Priyank Arora, and Anshul Verma. Document Summarization by Agglomerative nested clustering approach. *2016 IEEE International Conference on Advances in Electronics, Communication and Computer Technology, ICAECCT 2016*, pages 187–191, 2017.

[TPL14] Guanting Tang, Jian Pei, and Wo-Shun Luk. Email mining: tasks, common techniques, and tools. *Knowledge and Information Systems*, 41(1):1–31, oct 2014.

[Uni] The Pennsylvania State University. Agglomerative hierarchical clustering. https://onlinecourses.science.psu.edu/stat505/node/143/. Accessed: 2018-03-29.

[WLWK08] Ho Chung Wu, Robert Wing Pong Luk, Kam Fai Wong, and Kui Lam Kwok. Interpreting TF-IDF Term Weights As Making Relevance Decisions. *ACM Trans. Inf. Syst.*, 26(3):13:1—-13:37, jun 2008.

[XHLL16] Caiquan Xiong, Zhen Hua, Ke Lv, and Xuan Li. An Improved K-means Text Clustering Algorithm by Optimizing Initial Cluster Centers. In *2016 7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 265–268. IEEE, nov 2016.

[ZWM14] Mohammed J. Zaki and Jr. Wagner Meira. *Data Mining and Analysis: Fundamental Concepts and Algorithms*. Cambridge University Press, May 2014.

[ZXSX15] Dongwen Zhang, Hua Xu, Zengcai Su, and Yunfeng Xu. Chinese comments sentiment classification based on word2vec and SVMperf. *Expert Systems with Applications*, 42(4):1857–1863, mar 2015.