Improving telecommunication customer service through text analysis and categorization using semi-supervised learning

Maria Seabra Dourado Eusébio

Thesis to obtain the Master of Science Degree in

Mathematics and Applications

Supervisors: Prof. Maria do Rosário De Oliveira Silva
Prof. Isabel Maria Alves Rodrigues

September 2021

Abstract

With the increase of customers expectation, specially in telecommunication services, and being customer retention less expensive than customer acquisition, companies are investing more time and money on improving customer service, by automating internal processes and thus having a faster and more efficient troubleshooting process.

This work explores some methodologies that can be applied in the context of telecommunications and call centers, namely in the areas of text mining, Natural Language Processing, machine learning and Semi Supervised Learning algorithms, such as Label Propagation and Label Spreading, since the data provided is mainly unlabeled.

The primary objective was to classify each troubleshoot guide text according to each variable involved in the diagnosis of the recommendation system, as separate text categorization sub-problems, and from that build a new dataset with all the possible sequence of events and the correspondent troubleshoot guide. However, since the text categorization results obtained with the Semi-Supervised Learning algorithms were not as good as expected for some of the sub-problems, an alternative solution was found, consisting in, first, cluster the original labels into smaller sets and then classifying the texts into each of them. The solution obtained could be used to narrow down the huge amount of possibilities, helping the technical specialist in the task of associating the texts to the correct label in each sub-problem and, in that way, improving the existing recommendation system.

Keywords

Natural Language Processing; Text Mining; Text Categorization; Semi-Supervised Learning; Telecommunication Services

Resumo

Com o aumento das expectativas dos clientes, especialmente nos serviços de telecomunicações, e sendo a retenção de clientes menos dispendiosa que a aquisição dos mesmos, as empresas investem cada vez mais tempo e dinheiro na melhoria do serviço ao cliente, automatizando processos internos e obtendo um processo de resolução de problemas mais rápido e eficiente.

Este trabalho explora algumas metodologias que podem ser aplicadas no contexto das telecomunicações e call centers, nomeadamente nas áreas de mineração de texto, processamento de linguagem natural, aprendizagem automática e algoritmos de aprendizagem semi-supervisionada, tais como *Label Propagation* e *Label Spreading*, uma vez que os dados fornecidos são maioritariamente não etiquetados.

O objectivo principal seria classificar cada texto dos guias de resolução de problemas de acordo com cada variável envolvida no diagnóstico do sistema de recomendação existente, como sub-problemas de categorização de texto separados, e a partir daí construir uma nova base de dados com toda a sequência possível de eventos e o guia de resolução de problemas correspondente. Contudo, dado que os resultados da categorização de texto obtidos com os algoritmos de aprendizagem semi-supervisionada não foram tão bons quanto o esperado para alguns dos sub-problemas, foi encontrada uma solução alternativa, que consiste em, primeiro, agrupar as etiquetas originais em conjuntos mais pequenos e depois classificar os textos em cada um deles. A solução obtida poderá ser utilizada para reduzir a enorme quantidade de possibilidades, ajudando o técnico especialista na associação dos textos à etiqueta correcta em cada sub-problema, melhorarando o sistema de recomendações existente.

Palavras Chave

Processamento de Linguagem Natural; Mineração de Texto; Categorização de Texto; Aprendizagem Semi-supervisionada; Serviços de Telecomunicação

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

ML	Machine Learning
NLP	Natural Language Processing
POS	Part-of-Speech
SMOTE	Synthetic Minority Over-sampling Technique
BoW	Bag-of-Words
TF-IDF	Term Frequency-Inverse Document Frequency
CBOW	Continuous Bag-of-Words
SG	Skip-Gram
BERT	Bidirectional Encoder Representations from Transformers
MLM	Masked Language Model
NSP	Next Sentence Prediction
PCA	Principal Components Analysis
LSA	Latent Semantic Analysis
SVD	Singular Value Decomposition
LDA	Latent Dirichlet Allocation
MI	Mutual Information
NB	Naive Bayes
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
SSL	Semi-Supervised Learning
MAP	Maximum A Posteriori
EM	Expectation-Maximization
LP	Label Propagation
LS	Label Spreading

1 Introduction

Contents

In this chapter, the dissertation motivation is presented in Section 1.1. In Section 1.2, a summarized description of the problem at hand as well as the thesis objectives are explained. A literature review is discussed in Section 1.3. Lastly, the structure of this dissertation is described in Section 1.4.

1.1 Motivation

The continuing advances and innovation in technological services and devices, as well as the broad range of possibilities as a consequence of a growing competitivity between the companies that provide them, have increased the customers expectations and demands. Knowing that customer retention is largely less expensive than customer acquisition, companies are investing more time and money on improving customer service in order to improve the customer experience and satisfaction. In that sense, a growing number of businesses is trying to provide innovative strategies to enhance the operational performance of its call centers team members, reducing costs and increasing productivity and income. This can only be done by embracing the digital transformation that is quickly arising in all kind of businesses, finding the most appropriate tools to automate internal processes.

However, the real challenge is to find a balance between this automation and the human interaction, since many customers are happy to use automated services, but many others still want an element of human interaction, as they are emotional beings and to increase user engagement and satisfaction, each interaction must evoke positive emotions. Thus, the goal is to use an automated and dynamic solution that enhances the abilities of human agents on problem-solving and troubleshoot customer tech problems in a faster and more efficient way, so they can tackle more complex or creative tasks and, therefore, reducing complaints and improving customer engagement overall, which will definitely impact the business growth.

This work will focus on the call centers of the telecommunications industry, that consists of all the means that make communication possible on a worldwide scale, either through the phone, the internet or the television, with the use of cables or wirelessly. Since the early 2000s that this industry has become one of the most important and basic ones, with a rapid and continued growth in its technologies, but also a very complex one.

Troubleshooting telecommunications problems involves a quite complex process from the received call until the decision on the best solution. As stated above, there are many possible fields and each has its own specificities that, depending on the circumstances, may also vary. In order to diagnose the problem and obtain the best solution, the technical assistants should use the information given by the customer on the phone call, describing his complaint, along with some troubleshoot guides usually held in some database with texts explaining the process to follow after the diagnosis.

This is where Natural Language Processing (NLP) and Machine Learning (ML) techniques may prove useful, applying them to the available text data and hence helping the technical assistants in the troubleshooting of this telecommunication technical problems in service call centers. More specifically, text classification, where the main task is to assign to a text document one or more classes from a given set of possible classes.

A problem that arises is that obtaining a considerable large labeled dataset, necessary for the text classification task, is typically very expensive and requires a substantial human effort. Nonetheless, a solution to this issue, introduced in the 1960s, has recently become more popular and practically relevant: Semi-Supervised Learning (SSL). In SSL both labeled (usually scarce) and unlabeled observations are used and the general idea, when applied to the classification problem, is that the (majority) observations where the label is unknown can be used to improve the performance and to help in the classification process.

1.2 Problem Setting and Objectives

This research was held in partnership with the Center for Computational and Stochastic Mathematics (CEMAT) [2] and the consultant company Border Innovation [3], within the scope of the KEEN project. At first, the main goal of this project was to improve an already implemented recommendation system used in the technical support service of one of the Border Innovation's clients, a large telecommunication company, namely through developing automated solutions capable of increasing the effectiveness in the process of diagnosing the telecommunication issues that are reported to the company's technical assistants by its customers.

The process starts with some kind of failure in a device or service, that is then described by the customer when he calls to the technical assistant. Meanwhile, the recommendation system collects information about the customer's situation (a combination of variables), the *Scenario*, that will be used by the technical assistant during the call, to narrow down the existing possibilities. The technical assistant has also to select other variables in the recommendation system, based on the customer's complaints, namely the Service (Internet, Televisão/Television or Voz/Voice) in question and the Symptom related to the complaint. In each of them there are more specific options that should also be selected, Sub-service and Sub-symptom, respectively. With this information the recommendation system leads to the 3 most probable Causes, from an existing list of possible causes, for the customer's complaint.

After deciding which is the actual cause of the reported issue, the technical assistant follows some procedure according to the detected cause, in order to solve that specific problem. The list of all possible procedures (troubleshoot guides) is stored in a database out of the recommendation system, that has explanatory texts with the steps to follow after finishing the diagnosis in the recommendation system. The aim was then to bring together the information in the recommendation system and in these troubleshoot guides, so that the process could be even faster and more efficient.

For that purpose, Border Innovation provided a group of datasets with all the components involved in the process of several customer calls, labeled with the cause chosen by the technical assistant for each call, and also the links to the correspondent troubleshoot guide. As this is a sequential process – it starts with the Scenario, that includes the Service (`L0`), then the Sub-service (`L1`), the Symptom (`L2`), the Sub-symptom (`L3`) and finally the Cause (`CAUSE`), that will lead to a specific troubleshoot guide (`TEXT`) located in a certain link (`LINK`) – the first thought was to go through this path in an inverse way. In other words, the idea was to classify the `TEXT`s, according to each variable of the recommendation system, and then link them together, generating a new database with a list of all possible path combinations from `L0` to `TEXT`. In that way, there were five different classification sub-problems to solve, related to each variable of the recommendation system: `L0`, `L1`, `L2`, `L3` and `Cause`.

The provided data had some challenging issues, that are addressed in this research. Namely, being unstructured data (data that is not organized in a pre-defined manner), since it consists of text documents, and also the fact that most of the the observations (texts) have no associated labels (for each of the five sub-problems). Moreover, since the text data related to each `LINK` was obtained using a web crawler that only fetched the webpages text elements and not the text included in images, the quality of some of the texts was most likely compromised as some parts were missing. Thus, the main goals of this research are to use NLP techniques in order to solve the unstructured data issue and to be able to interpret and analyse the text data, as well as explore existing SSL models so that all the available data can be used. Moreover, it will be also addressed if this approach is in fact advantageous when compared to applying supervised learning techniques in the labeled, and smaller, part of the dataset. The objective is to use this knowledge and the best found approach to build five different models and with them construct a new database that could be added to the recommendation system in order to improve the troubleshoot process performed by the technical assistant, by indicating the correct troubleshoot guide to follow after the diagnosis that he made.

1.3 Literature Review

Since the areas of Text Mining and Text Categorization have become increasingly present in the reality of most companies and organizations, there are already several researches on these, with the aim of analyzing and processing unstructured data. The most common applications that have been studied are for example spam filtering, discerning e-mail spam messages from legitimate emails, and sentiment analysis, that is mostly used to identify positive and negative reviews usually based on social media platforms (e.g. Twitter).

Although the research in NLP has started around the 1940s, this field has only started to evolve and grow rapidly since the 1990s, having one of the first overall overviews of its definition, historical context, levels, approaches and applications in 2001, by Liddy [4]. More recently, in the Handbook of Learning Analytics, McNamara et al. [5] contribute with an introduction to the several NLP tools, used to understand discourse, as well as some applications of these tools for education. However, for a deep understanding of the various NLP tasks one should read Jurafsky and Martin's book [6]. It starts explaining the content at the word level, then syntax, semantics and finally pragmatics, including statistical methods with linguistic foundations and also computational models.

Kobayashi et al. [7], as well as Kowsari et al. [8], give an overview of all the steps involved in the text classification process, explaining some of the most important text mining techniques from machine learning and statistics, from the training data preparation to preprocessing, transformation, dimensionality reduction and finally the application of classification techniques and their validation.

Toman et al. [9] present a comparison between the usual word normalization approaches, stemming and lemmatization, explaining how lemmatization, particularly when dealing with highly inflected languages, is a very challenging task. In order to solve this issue, they introduce a new approach that transforms text into a language independent form, based on a multilingual database of words (EuroWordNet vocabulary), and they also try to understand the influence word normalization has on the text classification task.

After performing the preprocessing steps, it is necessary to transform the text data into numerical data so that it can be used in the classification algorithms. There are essentially two kinds of approaches to perform this transformation, by producing one number per each word (wordcount or bag-of-words) or by producing one vector per each word (word embedding). The former approach is the most simple one and it is referred in most of the researches in the NLP area (e.g. the first references of this Section) and the latter approach is a bit more complex, since it is based in neural network models. Mikolov et al. [10, 11] introduced the continuous bag-of-words and the continuous skip-gram models, that would later be used for the Word2Vec algorithm, and presented as well some extensions of that in order to improve the training speed and the quality of the vectors, by subsampling frequent words. Later, Alvarez and Bast [12] present an overview of the main word embedding algorithms, including Doc2Vec.

More recently, a great improvement was made in language representation, when Devlin et al. [13] introduced the Bidirectional Encoder Representations from Transformers (BERT) model, based on the notion of discourse coherence [14] and on a simple network architecture, the Transformer (presented by Vaswani et. al [1]), that makes use of attention mechanisms instead of recurrence or convolutions. A vast set of reviews have been made regarding this model and its performance [15, 16], finding, in particular, that despite its general good performance, BERT struggles with challenging inference and role-based event prediction, showing clear insensitivity to the contextual impacts of negation. Some of these surveys try to propose smaller and faster versions of this model, e.g. DistilBERT, proposed by

Sanh et al. [17]. One particularly interesting research for the context of this work is the one described by Souza et al. [18], where the BERT model is trained for the Brazilian Portuguese language.

One of the most problematic issues of text categorization is the high dimensionality of the feature space and its sparseness, since it consists of the unique terms (words or phrases) that occur in documents and that may correspond to tens or hundreds of thousands of terms, even for a moderate-sized documents collection. A simple approach, called feature selection, consists of removing the non-informative terms according to documents collection statistics. Yang and Pedersen [19], present a comparison between several feature selection methods in text categorization problems, namely Chi-Square and Mutual Information (MI), whereas Vergara and Estévez [20] focus on the latter.

Another possible approach, feature extraction, consists of constructing new features, by combining lower level features (words) into higher level orthogonal dimensions. According to Johnson et al. [21], Principal Components Analysis (PCA) is the most common technique and it is used in problems of several areas (not just text categorization), however, other methods have been studied in the text categorization problems, such as Latent Semantic Analysis (LSA), that tries to solve the polysemy issue (words that have multiple meanings), as it is well explained in Rosario's overview [22]. A more complex technique is later introduced by Blei et al. [23], the Latent Dirichlet Allocation (LDA) model, that is then complemented with Gibbs Sampling in order to approximate the posterior distribution that cannot be directly computed, as it is explained by Liu [24] and also by other authors [25–28]. More recently, Norsten [29] applies this method to sentiment analysis, using Twitter data.

A comparison between all the previous methods can be found in some surveys [30, 31], but not quite consistent with the results, which is most likely related to a dependence on the application and on the kind of data at hand.

With the data prepared, there can be then applied the text categorization techniques. The goal is to classify the documents into a set of categories. Ideally, this would be performed by supervised learning techniques, such as the Naive Bayes (NB) classifier, explained for example by Berrar [32], that despite of its conditional independence assumption that is usually not satisfied in real applications, has proven to be one of the most efficient algorithms for ML. Zhang [33] tries to infer the reason for this unexpected good performance, concluding that no matter how strong the dependencies among attributes are, NB can still perform well if the dependencies distribute evenly in classes, or if they cancel each other out.

However, this requires an extra human effort in order to pre-define the categories and assign them to the documents of the training set, an issue that the unsupervised leaning techniques try to overcome. This approach applies clustering algorithms to the dataset, such as K-means, presented by Likas, Vlassis and Verbeek [34], a point-based clustering method that starts with the initial cluster centers at arbitrary positions and proceeds by moving them at each step in order to minimize the clustering error, or Density-Based Spatial Clustering of Applications with Noise (DBSCAN), introduced by [35], relying on a density-based notion of clusters with the aim of discovering clusters of arbitrary shape. These techniques, when applied to text data, are based in document similarity, that can be measured by, for example, cosine distance, Euclidean distance or the manhattan distance, as it is explained in various surveys [36–38]. A comparison between these and other clustering algorithms, with advantages, disadvantages and applications for each of them, can be found in [39].

Lee and Yang [40] provide a comparison between supervised and unsupervised learning approaches for multilingual text categorization, aiming to make use of the unsupervised approaches as a first hint to enhance the task of pre-defining categories and constructing the labeled training set. In fact, a similar approach was previously introduced by Ko and Seo [41], in which an unsupervised method divides the documents into sentences and then categorizes each of them based on keyword lists of each category and sentence similarity measure, resulting in a set of categorized sentences that can be then used for training. This idea of combining both supervised and unsupervised learning resulted in the SSL approach.

SSL is actually more practical and useful for solving the problems in most of the projects, as most of the data are unlabeled but some data are labeled, since labeled data are very hard to access but unlabeled data are usually easily collected and accessed. Chapelle, Schölkopf and Zien [42] gather together a collection of articles from many authors that have been researching in this field, forming a rather complete overview of the existing methods in the SSL approach, categorizing them in self-labeled methods, generative models, low-density separation algorithms, graph-based methods and explaining their particularities and applications. The most simple ones are the self-labeled techniques, described in [43] by Triguero et al. Another technique also discussed in this approach is the iterative computation of maximum-likelihood estimates when the observations are considered incomplete, where in each iteration an expectation step is followed by a maximization step and is therefore called Expectation-Maximization (EM) algorithm, as presented by Dempster et al. [44]. A review of these several SSL models is later presented by Bagherzadeh and Asil [45], with an approach to deep learning.

Bocancea [46] explores how unlabeled data can improve supervised learning classifiers in all contexts, for both scarce to abundant label situations, overcoming issues related to selection bias, noise and insufficient data.

There exist already numerous possible ways of evaluating the performance of the algorithms presented above. A wide range of researches has already presented a comparison between the existent performance measures for classification [47, 48], with Lewis [49] being one of the first ones to address this to text classification. Also, for clustering algorithms, average silhouette is introduced by Rousseeuw [50], based on the comparison of each cluster tightness and separation. An interesting research was made by Amigó et al. [51], where a few intuitive formal constraints on text clustering algorithms evaluation metrics are defined, leading to which aspects of the quality of a clustering are captured by each metric.

Regarding the measurement of good generalization performance of a classification model, a few data splitting strategies, crucial for model validation, are described by Xu and Goodacre [52]. On the other hand, Berrar [53] focuses on the cross-validation method, introducing its most common techniques, such as k-fold cross-validation.

Moreover, a common issue in multi-class problems are the imbalanced datasets. The evaluation metrics should be carefully chosen for this kind of datastes and specific metrics are described by Bekkar et al. [54] and also Branco et al. [55].

There are several possible approaches in order to deal with the imbalanced datasets and the most common, such as over sampling, under sampling and Synthetic Minority Over-sampling Technique (SMOTE) consist of resampling the dataset. These techniques were already addressed by various authors [56–58].

On the other hand, Osborne and Overbay [59] summarize several possible causes of extreme scores in a data set, ways of detecting them and whether they should be removed or not. Moreover, they explore how a small proportion of outliers can greatly affect even simple analyses.

1.4 Thesis Outline

This dissertation is divided in six chapters, including this introductory one. In Chapter 2 an introduction to the background results related to the problem at hand is made and the methods and techniques used in this study are explained in detail, starting with the concept of text mining and then some approaches involved in the text data preparation (text preprocessing, text representation and dimensionality reduction). Chapter 3 continues to introduce theoretical concepts, namely the models used for text categorization and also some evaluation metrics. Then, Chapter 4 explains the particularities of the explored dataset and how it is organized, as well as the process of data preparation until the implementation of the models, their specifications and how they were applied to it. In Chapter 5, each model is evaluated over different metrics, for each of the five sub-problems and according to each scenario (Semi-Supervised Learning, Clustering then Semi-Supervised Learning or Clustering) explained in the previous Chapter, presenting the results and a small comparison. Finally, Chapter 6 presents the closing remarks of this thesis and also some suggestions for future work.

2

Text Mining

Contents

This chapter provides the theoretical background necessary to understand the methods applied for the text analytics in this thesis. It starts by explaining the context of the problem, introducing the concept of text mining and NLP. Then, a theoretical introduction to the methods and algorithms used in each step of the process, from the text data preparation, including preprocessing, transformation, and dimensionality reduction is made.

2.1 Text Mining

According to [60], text mining is similar to data mining, in the sense that both try to explore and identify interesting patterns from the given data and then extract relevant information from it, using some analysis methodologies. The difference is that in text mining the data comes from document collections, called corpus, consisting of (usually) unstructured textual data. Therefore, it is necessary to transform text into structured data and there is where the NLP appears.

NLP is described in [5] as the analysis of the natural human language using computers. It basically consists in making use of some ML techniques with a view to automate the comprehension, analysis and generation of natural language. Moreover, there are several fields [61] where NLP can be applied such as: natural language text processing and summarization, machine translation, multilingual and cross language information retrieval, speech recognition and artificial intelligence. In particular, for natural language text processing there are different "levels" of language that can be considered by an NLP system, for instance, morphology, lexical and sintactic [4].

2.2 Text Preprocessing

First of all, it is important to understand that whilst working with texts, the dataset that will be analysed has its set of observations as a set of documents (document collection or corpus) and its features are usually words characteristics deriving from each document.

Furthermore, the effectiveness and quality of the text mining operations depend deeply on the data preprocessing methodologies [60]. As a matter of fact, these preprocessing techniques are used to infer or extract (or both) structured data from the original text documents. In the following subsections some of these techniques are detailed.

2.2.1 Tokenization

Typically, the first step in text preprocessing is to apply tokenization, resulting in a set of individual terms instead of the original continuous text [7]. The generated terms are called tokens and, although they are usually words, they can also be punctuation marks, numbers, symbols, compound expressions. If the token is a set of n consecutive expressions it is called an $n-$gram [62]. Moreover, these terms are typically separated by white spaces or punctuation marks in the text.

2.2.2 Stop Words

The next step is to remove the most common words since they add no (or very low) information and therefore are irrelevant. These are called stopwords and, in Portuguese, they can be, for example, prepositions, conjunctions, articles or pronouns, such as “e”, “que”, “um”, “uma”, etc. [62].

At this stage it is also common to remove other kinds of tokens, for instance punctuation and numbers, unless they are relevant for the problem at stake (e.g. exclamation marks in sentiment analysis, [7]).

Additionally, spelling correction is almost always applied in order to identify and rectify some possible orthographical errors, typos or even abbreviations, using a certain dictionary.

2.2.3 Part-of-Speech Tagging

Part-of-Speech (POS) tagging consists in associating words with a suitable category, based on their function (and position) in the sentence they appear in [60]. This category, called POS tagg, is somehow a word feature that holds information about the semantic content (meaning) of that word.

The most commonly used tags are Article, Noun, Verb, Adjective, Preposition, Number and Proper Noun, although they may be considered a lot more word classes, up to 146 [6, 63].

Besides the semantic analysis, a morphological analysis can also be included in some POS taggers, that is, they may also include other word characteristics, as the ones introduced in the Subsection 2.2.4.

2.2.4 Lemmatization and Stemming

There are two main approaches to word normalization: lemmatization and stemming. Although they seem similar, there exist some important differences between these two techniques.

The process of obtaining the radical or stem of each word is called stemming and consists in removing suffixes and/or prefixes that do not correspond to the base form of the word, obtaining an approximation of this form [7, 9]. The main idea is that words with similar base forms have similar meaning and so they are processed as the same word (the stem).

On the other hand, in order to recover the base form itself, lemmatization should be applied. The canonical form (base form) of a word is called lemma and it is obtained by removing the suffix/prefix of a word completely or replacing it with a different one [9]. In Portuguese, the lemma of a verb is usually the verb in the infinity form or, in the case of nouns and adjectives, the lemma is the word in singular and masculine form [64]. For instance, for the word “gostei” (that is a verb), the lemma is “gostar” but the stem is “gost” and the word “esperta” (that is a noun) as the stem “espert” and lemma “esperto” [64].

It is important to notice that lemmatization is more challenging than stemming, specially when processing highly inflected [1] natural languages like, for example, Portuguese, which may overrule its benefits. However, in some cases it can be beneficial, when the base form of the word is required [7, 9].

Moreover, both stemming and lemmatization usually result in a loss of inflection information in words, such as tense, gender, and voice. Since this information is sometimes important in some applications (e.g. most negative reviews are written in the past tense, according to [7, 65] , it may be wiser not to apply them in text preprocessing. Therefore, word normalization should be applied only when it increases classification efficiency (reducing the number of terms) and when no meaningful degradation results from it [7].

2.2.5 Imbalanced Data Sets

An imbalanced data set is one with class imbalance, that is, when there are much more observations labeled as some of the classes than as the others. This leads to some biased estimates of standard evaluation metrics, as typically they tend to give more importance to the majority classes and ignore the minority ones [57, 66]. Some alternatives are going to be discussed in Section 3.5.

Nevertheless, there are a set of approaches that can be applied to an imbalanced dataset in an attempt to solve this issue and basically consist of re-sampling the original dataset. This can be performed either by removing some of the observations associated with the majority classes, undersampling, or by generating observations from the minority classes, oversampling [56]. However, it has been observed that oversampling the minority classes results in an increase of the sample size and, consequently, it is necessary more time to train the model. On the other hand, although undersampling the majority classes could lead to missing out on some actual useful data, it actually leads to some effective results [57, 58].

To overcome this problem, Chawla et al. [56] proposed an oversampling method, called SMOTE. In this case, new "synthetic" observations from the minority classes are created, by interpolating between some of its nearest neighbors (randomly chosen) or, in other words, by adding these new observations along the lines formed between the minority samples and its neighbors, as shown in Figure 2.1. The last authors went even further, by smoting the minority class and then undersampling the majority class with the aim of turning the initial bias of the classifier towards the majority classes in the favor of the minority classes. However, the benefits of this strategy strongly depend on the minority sample size.

2.2.6 Outlier Detection

Another possible pre-processing step is the outlier removal. This step consists of cleaning the data by removing some atypical observations, called outliers, that may have an excessive impact on the classical

[1] inflection refers to the changes made in the base form of a word in order to show its grammatical relations

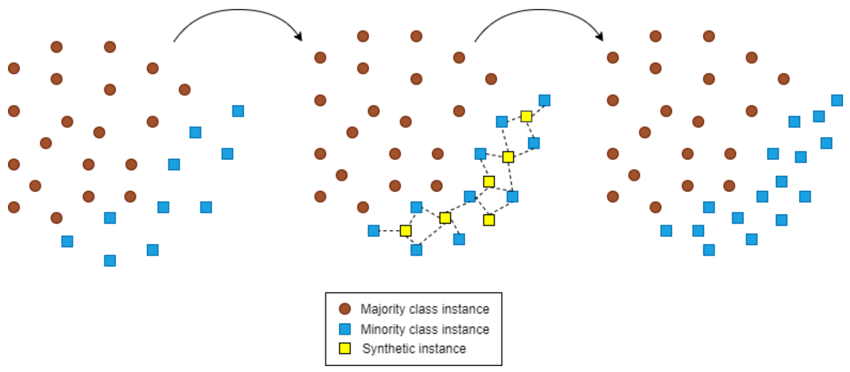

Figure 2.1: Example on how new synthetic observations are created in the SMOTE method.

estimates. These outliers can derive from unintentional (e.g. misinterpretation from the person who fills out the survey that will result in the data) or intentional errors (e.g. when the person who is fulfilling the survey purposefully reports incorrect data) or they can even come from an outside the scope population.

The outliers detection can be performed by parametric or non-parametric techniques. While, on one hand, the parametric methods try to fit the data distribution and then identify which observations are not likely to occur according to that layout, on the other hand, the non-parametric methods do not require any distribution assumption, that may be hard to be verified in practice. Some examples of non-parametric methods will be further explained in Section 3.3.

Removing outliers is not always the best practice. It could be the case that an observation that seems to be an outlier is actually a legitimate observation that carries relevant information (sometimes rare translates into very important) and there is no straightforward rules to accurately decide whether or not an observation is an outlier [59].

2.3 Text Representation

Since the known learning algorithms are developed to process numerical data and not words, it is necessary to transform the texts into structured data in order to apply these algorithms to them. Thus, the texts are usually represented by feature vectors, in $\mathbb{R}^n$, being each text "a sequence of features and their weights" [60].

2.3.1 Bag-of-Words

The simplest and most common way of representing these feature vectors is through a Bag-of-Words (BoW). It can be seen has a document term matrix, where each row represents a document, each column represents a feature (term, that is usually associated to a specific word) and each matrix entry corresponds to the weight of the term in that document [7], typically term frequency or counts (number of times each word occurs in the document).

The name BoW is related to the fact that each document is represented by a set of words that are not ordered. Hence, this set of words cannot be seen as a sentence, since “the semantic relationship between these words is ignored” [8], although the multiplicity is taken into account through word frequency. Another issue of this model is that higher weights are assign to the most common words, that usually are not that important in the text.

2.3.2 Term Frequency - Inverse Document Frequency

In order to solve the last issue explained above, there is an alternative weighting scheme, the Term Frequency-Inverse Document Frequency (TF-IDF). This model takes into account not only the word counts in the document but also the frequency of that word in the whole document collection, penalizing the most common words. The TF-IDF weight of word w in document d, $W_{TF-IDF}(w,d)$, is generally given by:

$$W_{TF-IDF}(w,d) = TF(w,d) \times \log\left(\frac{m}{df(w)}\right) \tag{2.1}$$

where $TF(w,d)$ is the frequency of word w in the document d, $df(w)$ is the number of documents that contain word w and m is the total number of documents.

Although TF-IDF can fix some of the BoW drawbacks, it is not yet able to include the similarity between words in each document, since words still have no meaning and no relation between them [8].

2.3.3 Word Embeddings

Word embedding is a more complex technique, where the meaning of words is represented by vectors of real numbers in a high dimensional space. The main assumption is that words that occur in similar contexts, or that are related to each other, have similar meanings and so their embeddings are placed close to each other [12].

Moreover, according to [12], this technique includes the factorization, in two separate embedding spaces, of a word-word matrix. Each entry of this matrix correspond to a distance metric, such as co-occurrence counts or point-wise mutual information, calculated between any two words.

The factorization is made through an analysis to the text or document collection, with a fixed sized window. This window is characterized by the target word and some of its neighboring words, called the context. In the factorization, the target embedding is initialized in a matrix that contains the target word embeddings, whereas the context embbeding is initialized in a matrix containing the representations used for the context words in that window. Hence, the main objective is to minimize the distance between words and their contexts, by performing stochastic gradient descent through each consecutive window of the text.

There are several methods for word embedding being Word2Vec one of the most common algorithms from the traditional word embedding techniques. In this kind of technique, a (shallow) neural network is trained in order to learn and predict the vector representation of each word and consists of only an input layer, a projection layer and an output layer [67]. The simplicity of this models, due to the existence of only one hidden layer, enables the opportunity of being trained on a much higher quantity of data (more words) in a more efficient way [10].

2.3.3.A Word2Vec

There are two main neural network architectures that can be used by Word2Vec: Continuous Bag-of-Words (CBOW) and continuous Skip-Gram (SG). Whereas CBOW predicts a target word from one or more context words, the SG does exactly the opposite, predicting one or more context words from a target word [68], as described in Figure 2.2.

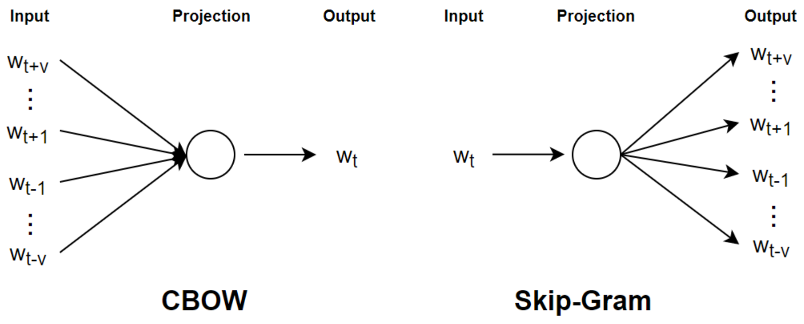

Figure 2.2: CBOW and Skip-gram model architectures.

SG model consists of a neural network with just an input layer, a projection layer and an output layer. This model, introduced by [11], learns the vector representation of each word with the objective of maximizing the average log probability

$$\frac{1}{n}\sum_{t=1}^{n}\sum_{-v\leq j\leq v, j\neq 0} log\ p(w_{t+j}|w_t) \tag{2.2}$$

that is, it maximizes the log probability of neighboring words in a corpus, where n is the total number of words used in the training (it is assumed to be equal to the total number of words in the vocabulary) and v corresponds to the size of the training context (neighbors of the word w_t). In other words, the model predicts words in the neighborhood of the target word, w_t.

The probability in (2.2) is defined by the softmax function:

$$p(w_{t+j}|w_t) = \frac{exp({v'_{w_{t+j}}}^T v_{w_t})}{\sum_{i=1}^{n} exp({v'_{wi}}^T v_{w_t})} \tag{2.3}$$

where v_{w_t} is the input vector representation of the target word w_t, ${v_{w_{t+j}}}'$ is the output vector representation of w_{t+j} and n corresponds to the total number of words in the vocabulary. Since this is associated to a quite high computational cost (proportional to n that is usually very large, between 10^5 and 10^7 words), the softmax function must be approximated in order to be used to compute the probability estimates. Hence, this probability can be efficiently calculated using hierarchical softmax or negative sampling [11].

Hierarchical softmax uses a binary tree to represent the output layer and so, instead of evaluating n output nodes in the neural network it evaluates only around $log_2(n)$ nodes. Negative sampling is based on Noise Contrastive Estimation (introduced by [69]) and tries to differentiate the target output word, w_{t+j}, from noise using logistic regression. However, this differentiation is performed by only looking at samples (with k negative samples each, being k a value between 5 and 20 for small training datasets or between 2 and 5 for larger ones, [11]) instead of looking at the numerical probabilities of the noise distribution.

Even though Word2Vec technique includes the similarity and relation between words, it does not take into account the actual context of the word, since each word has only one vector representation. That is, even if the word is used in two different sentences with two different contexts, it has always the same vector representation. Thus, Word2Vec cannot handle polysemous words.

2.3.3.B BERT

Fortunately, later there were introduced the contextual embedding methods, that use the whole sentence as an input, instead of just one word. Therefore, these techniques can learn more than one vector representation for each word and thus fix the polysemy problem. However, in order to have the sentence and its context as the input and to generate the embeddings according to it, these techniques need a pre-trained model.

BERT, is one of these language representation models and was introduced by [13]. Its great particularity is that it uses a Masked Language Model (MLM) and so both left and right (bidirectional) context of each word is taken into consideration, making it possible to pre-train a deep bidirectional Transformer. Basically, a MLM, as the name suggests, masks some of the input words and then tries to predict the original vocabulary id of each of those masked words, using only their context [13].

On the other hand, a Transformer is a simple network architecture that, instead of using recurrence or convolutions that are usually more complex and are associated with memory problems, uses attention mechanisms [1], more specifically self-attention. Simply speaking, attention is a mechanism that takes two sequences (in this specific case, sentences) and builds a matrix where the rows correspond to the words of one of the sentences and the columns correspond to the words of the other. It then tries to identify some relevant context by matching each pair of words and attributing a certain weight to it, through a softmax function [1]. In particular, self-attention uses only one sentence and then relates different positions of that sentence in order to compute its representation. Figure 2.3 better illustrates this process.

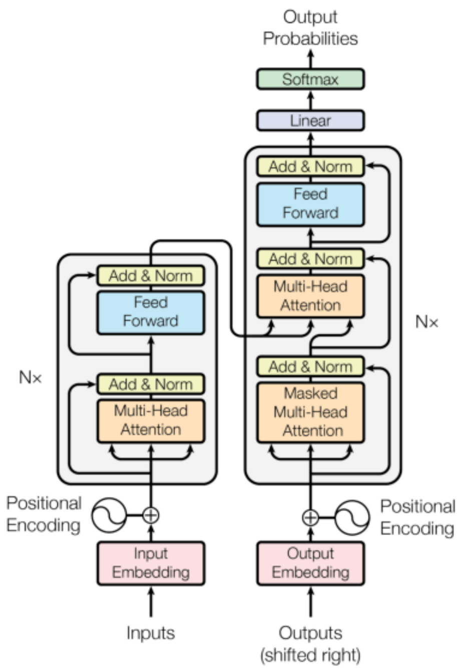

Figure 2.3: Transformer model architecture (image taken from [1]).

Getting back to BERT, it consists of two main steps: pre-training and fine-tuning. The pre-training step, in turn, uses two unsupervised tasks, namely MLM and Next Sentence Prediction (NSP). MLM, as already explained, masks some of the input tokens randomly and then predicts them, allowing BERT to train a deep bidirectional representation. This is a necessary task since, in a standard conditional language model, it would not be possible to use bidirectional conditioning as it would be trivial to the model to predict each word in a multi-layered context, considering that each word would be capable to indirectly "see itself" [13].

On the other hand, NSP task is important to understand the relationship between sentences, that is the context, since it is usually not directly captured by language modeling. This task is based on coherence relations [14]. The idea is that any two sentences from the same paragraph should demonstrate some coherence and usually two consecutive sentences are more coherent than two sentences that are more distant from each other. With this in mind, the NSP task takes the first three sentences of a paragraph and considers a set of five sentences from a later part of that paragraph as candidates. The goal is to decide which of these five candidates immediately follows the first three sentences.

In the fine-tuning step, a simple classification layer is added to the pre-trained model and then the parameters are fine-tuned all together on a further task. Thanks to the self-attention mechanism in the Transformer, BERT is able to model several further tasks and thus fine-tuning is quite straightforward. For each task, the specific inputs and outputs from that task are inserted into BERT and then all the parameters are fine-tuned end-to-end. Therefore, the fine-tuning step is much less expensive and time-consuming than the pre-training step and that is why, when using BERT for text representation, the pre-training step is not performed and instead an existent model is applied.

Some of the drawbacks of BERT are, for instance, the fact that it is not able to understand the relation between its known words, the results are not sufficiently good when trying to perform some pragmatic inference and it struggles with negative expressions [15, 16].

2.4 Dimensionality Reduction

As seen before, when dealing with text data the observations correspond to the documents or sentences and the unique words that occur in those documents, also known as tokens, correspond to the features. An issue that immediately stands out is the fact that the number of features may go up to hundreds or even thousands of terms, which are way to high dimensions for some of the classification algorithms to handle.

This is associated with the curse of dimensionality, that says that the number of observations needed to estimate an arbitrary function with a determined accuracy grows exponentially with respect to the number of features of that function [70] and that is essentially caused by the sparsity of data in high-dimensional spaces. In order to solve this problem, there were introduced several dimensionality reduction algorithms, that are typically divided into two types of classes: feature extraction and feature selection. This is a common step in text science.

2.4.1 Feature Extraction

One way of reducing the dimensions of the feature space, n, is to build a new set of features, with much smaller dimensions, k, where $k \ll n$, based on the original feature set [60].

2.4.1.A Principal Components Analysis

PCA is one of the most common techniques for dimensionality reduction. It essentially projects the original feature space into a sub-space, where the new features are uncorrelated and (linear) combinations of the original features [30].

Throughout this process, there are some steps that need to be followed, such as compute the sample mean vector and sample covariance matrix, S, of the original variables and then find the eigenvectors of S. These eigenvectors, extracted from the sample covariance matrix, are called the principal components , are uncorrelated and can be defined as a linear combination of the original features. In this way, the principal components are then sorted according to the corresponding eigenvalues, in decreasing order, that measures the $\%$ of the total sample variance explained by each principal component. The ones with the k highest eigenvalues are chosen and define the new k features (principal components). The other eigenvectors are simply discarded.

In order to better understand this, it is useful to look at the matrix representations. Consider the matrix $X_{m\times n}$, where m is the number of documents (observations) and n is the number of word tokens (from the original feature set). The first step is to compute the sample covariance matrix of X, $S_{n\times n}$, as follows [21]:

- Compute the column mean vector $\bar{x} \in \mathbb{R}^n$, where $\bar{x_j} = \frac{1}{m}\sum_{i=1}^{m} x_{ij}$ for $j = 1, ..., n$.
- Subtract $\bar{x}$ from each row of X, obtaining $\tilde{X}$ where $\tilde{X}_{ij} = X_{ij} - \bar{x_j}$, for $i = 1, ..., m$ and $j = 1, ..., n$.
- Calculate the sample covariance matrix $S = \frac{\tilde{X}^T\tilde{X}}{m-1} = \begin{bmatrix} \hat{\sigma}^2_{11} & & & \\ \hat{\sigma}^2_{21} & \hat{\sigma}^2_{22} & & \\ \vdots & \vdots & \ddots & \\ \hat{\sigma}^2_{n1} & \hat{\sigma}^2_{n2} & \dots & \hat{\sigma}^2_{nn} \end{bmatrix}$.

where $\hat{\sigma}^2_{ij}$ represents the sample covariance between the word tokens i and j.

An interesting point to notice is that, since the sample covariance matrix is always symmetric and semi positive definite, then the eigenvalues will be always real and non-negative.

Next, the idea is to compute the matrix W, with dimensions $n \times n$ and where the columns correspond to the n eigenvectors of S. Finally, after sorting the correspondent eigenvalues in descending order and choosing the k highest ones, it can be defined a $n \times k$ matrix, W_k, where the columns relate to the respective k eigenvectors. Thus, the matrix that describes the new and reduced feature set is given by $X_k = XW_k$.

Note that the computations of the matrix S are quite expensive, hence, an alternative to speed up these computations is to use Singular Value Decomposition (SVD). But how thus it work for the square matrix S? For the sake of simplicity, $\tilde{X}$ will be written as X in the next few lines.

The SVD of X is defined by $X = U\Sigma V^T$ where Σ is a $m \times n$ diagonal matrix with the singular values of X, V is a $n \times n$ matrix with the right singular vectors of X and U is a $m \times m$ matrix with the left singular vectors of X [71]. Therefore, $X^TX = (U\Sigma V^T)^T U\Sigma V^T = V\Sigma^T\Sigma V^T$, where:

- $\Sigma^T\Sigma$ is a square diagonal matrix in which the diagonal entries are equal to $\hat{\sigma}^2_i$ (the square of the singular values of X or also the i-th eigenvalue of X^TX), with $i = 1, ..., n$.

- V is a $n \times n$ matrix in which the columns correspond to the right singular vectors of X and also the eigenvectors of $X^T X$.

Thus, it can be infered that $W = V$ and therefore the matrix $S = X^T X$ does not need to be calculated in order to get its eigenvectors.

Also, note that this is exactly the same as computing the eigenvalue decomposition of $S = X^T X$. That is, since S is a symmetric and real square matrix, it can be decomposed as $S = VDV^T$, where V is a $n \times n$ matrix in which the columns correspond to the eigenvectors of S and D is a square diagonal matrix with the diagonal entries equal to the eigenvalues of S, $\lambda_i = \hat{\sigma}_i^2$, in nonincreasing order.

One of the main problems of this algorithm is that it is scale sensitive, and that is why the standardization of data is so important. Moreover, as the new features (principal components) are linear combinations of the original ones, they turn out to be not as readable and interpretable as the original ones. The selection of the number of principal components should be made with care, as it may miss some relevant information from the original features [21].

2.4.1.B Latent Semantic Analysis

LSA, first introduced by [72], is another dimensionality reduction technique, essentially used in text classification, that relates the words by some latent concepts in the text. It generates a set of concepts, smaller than the original word feature set, and then relates to the words and the documents [30]. This approach is very similar to PCA and uses SVD to obtain these concepts, that is, the reduced dimensions of the original matrix, X with dimension $m \times n$, where m is the number of documents and n is the number of words. The difference will be that LSA uses the eigenvectors of X whereas PCA uses the eigenvectors of the sample covariance matrix of X, S [31]. In this way, LSA is able to work out the problem of multiple words with similar meanings (synonyms) and words with more than one meaning (polysemy).

However, this method has some drawbacks as well [22]. One of these issues is related with data storage, since when using SVD the sparseness of matrices is lost, which is computationally expensive. Another problem is that LSA is supposed to be applied to normally-distributed data, which is not the case in term-by-document matrices, described by count data. As in PCA, the selection of the number latent concepts should be made with care, as it may miss relevant information from the original features [21].

2.4.1.C Latent Dirichlet Allocation

LDA is a more complex dimensionality reduction technique, since it is primarily a generative probabilistic model of a corpus, that is, it can generate new documents based on the distributions over latent topics and over words [23]. The main idea is that the documents are characterized by a set of topics and each of those topics is represented as a set of words, through probability distributions. In this way, the topics will work as “bridges” between the documents and the words.

Hence, the objective of LDA is to find which topics could represent, in a reduced space, the documents based on its words, while keeping the statistical dependencies [28]. Moreover, this model assumes that all documents in a corpus are represented by the same set of topics, but with different proportions in each one of them.

For the sake of simplicity, the generative process will not be explained in detail and the focus will be in the model inference and parameter estimation. However, it is useful to know that the generative process is described as follows [27]:

1. For $t = 1, ..., k$:
   (a) $\phi^{(t)} \sim$ *Dirichlet*(β).
2. For each document $d \in \mathbf{D}$:
   (a) $\theta_d \sim$ *Dirichlet*(α).
   (b) For each word $w_i \in d$:
      i. $z_i \sim$ *Multinomial*(θ_d);
      ii. $w_i \sim$ *Multinomial*$(\phi^{(z_i)})$.

where k is the number of latent topics, $\mathbf{D} = \{d_1, ..., d_m\}$ is the corpus (collection of documents), $\phi^{(t)}$ represents the t-th topic distribution, θ_d represents the distribution of document d over the topics, z_i is the topic index for the word w_i and α and β are hyperparameters for the symmetric Dirichlet distributions which lead to the Dirichlet-multinomial distributions. A graphical representation of the LDA generative process can be found in Figure 2.4. Here, the rectangles correspond to loops, where the outer one represent the documents and the inner one represent the repeated choice of topics and words within a document [28].

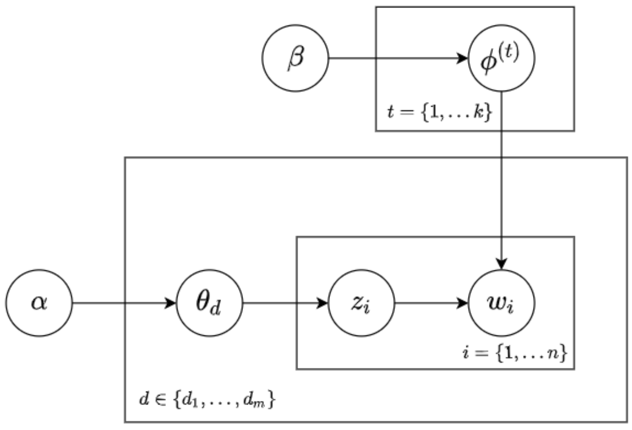

Figure 2.4: Graphical representation of LDA generative process.

This results in a joint distribution given by:

$$p(w, z, \theta, \phi|\alpha, \beta) = p(\phi|\beta)p(\theta|\alpha)p(z|\theta)p(w|\phi_z) \tag{2.4}$$

According to [29], there are some assumptions on LDA model that need to be considered:

- The order of the words that represent each document does not matter;
- Words that appear more than 80% to 90% in all documents do not convey meaningful information (usually stop words) about the topics and so can be discarded;
- The number of topics, k, is pre-defined.

Therefore, in the inference model the words are divided into two different types. On one hand, there are the words that belong to a document and that are already known and, on the other hand, there are the words that will represent a certain topic (after being assign to it). More precisely, the latter words are associated with a probability of belonging to that topic, which needs to be calculated for each of those words.

Note that the inference model consists of reversing the defined generative process in order to learn the posterior distributions of the latent topics in the model given by the observed words:

$$p(z, \theta, \phi|w, \alpha, \beta) = \frac{p(w, z, \theta, \phi|\alpha, \beta)}{p(w|\alpha, \beta)} \tag{2.5}$$

However, the term $p(w|\alpha, \beta)$ cannot be computed exactly and so an approximate estimate needs to be calculated. This is where Gibbs Sampling comes in. Not going into the details, what Gibbs Sampling does is constructing a Markov chain that has the target posterior distribution as its stationary distribution. That is, after a certain number of iterations through this chain, sampling from the conditional distributions of the variables of the posterior should converge to the sampling from the desired posterior [27].

Another algorithm, collapsed Gibbs sampler, can be derived from the latter, using the relevant parts for LDA, that are the latent document-topic portions, θ_d, the topic-word distributions, $\phi^{(z_i)}$, and the topic index assignments for each word, z_i. This algorithm computes the posterior $p(z_i|z_{-i}, \alpha, \beta, w)$, up to a constant, which defines the probability of a topic, with index z_i, being assigned to the word w_i, given all other topic assignments to all other words. The notation z_{-i} refers to all the topic allocations except for z_i. After some calculations, applying conditional probability rules and the chain rule, the Gibbs sampling equation for LDA is obtained and therefore the posterior (all details are explained in [27]).

In order to understand better the theory, it is probably useful to see what actual happens when implementing this LDA collapsed Gibbs sampler. Consider a $m \times n$ matrix X, where m is the number of documents and n is the number of words. The goal is to obtain the probability of a word w belonging to a topic t and so the following algorithm should be applied [29]:

1. Go through each document d and randomly assign each word in the document to one of k topics (k is chosen beforehand).

2. Calculate the number of words assigned to topic t in document d, $n_{d,t}$, for each document and through all k topics, getting a $m \times k$ count matrix C^{MK}.

3. Calculate the number of times word w is assigned to topic t, $n_{t,w}$, for each word and through all k topics, getting a $n \times k$ count matrix C^{NK}.

4. For each document, d, go through each word, w, and compute:

   (a) p(topic t| document d) $= \frac{C^{MK}_{d,t}+\alpha}{\sum_{j=1}^{k} C^{MK}_{d,j}+k\alpha}$;

   (b) p(word w| topic t) $= \frac{C^{NK}_{w,t}+\beta}{\sum_{i=1}^{n} C^{NK}_{i,t}+n\beta}$;

   (c) Compute p(word w belonging to topic t) $=$ p(topic t| document d) $\times$ p(word w| topic t);

   (d) Update the C^{NK} and C^{MK} count matrices with the new sampled topic t for word w.

5. Go back to $1.$ and repeat all the steps until convergence.

Note that the probability in (a) corresponds to the proportion of words in document d that are currently assigned to topic t. It tries to capture how many words belong to the topic t for a given document d, excluding the current word. Hence, if a lot of words from d belong to topic t, it is more likely that word w belongs to topic t. In the same way, the probability in (b) corresponds to the proportion of documents assigned to the topic t through the word w. It tries to capture how many documents were assigned to topic t because of word w.

At the end, for each document, there will be a vector of length k which represents the probability distribution of the document over the k topics. Thus, appending such vectors for all the documents, it is possible to obtain a $m \times k$ matrix, X_k.

One of the main advantages [24] of LDA is the fact that it is a probabilistic model and so it produces interpretable topics and can be used to estimate model parameters. Moreover, it generally handles well large corpus since the number of parameters in the model is not related with the number of documents.

In contrast, LDA does not capture possible correlations between the topics. As before, the number of topics needs to be settled beforehand [25, 26].

2.4.2 Feature Selection

Another approach to dimensionality reduction in text data is to remove irrelevant words and it is called feature selection. Actually, a large part of the words that occur in this text data are irrelevant for classification tasks and dropping them from the feature space causes no harm to the classifier performance and, in fact, leads to some noise removal, which may improve its performance [60].

2.4.2.A Chi-Square

The Chi-Square (or χ^2) is one of the most popular feature selection methods that are used in text classification. It is a statistical feature selection method, based on the χ^2 distribution. Thus, it tests the independence of two events, occurrence of the word and occurrence of the class, measuring the lack of independence between them.

In other words, the higher the Chi-Square value, the more dependent on the class the word is and so it is selected for model training and if the word and the class are independent the Chi-Square value is small (near to zero). It is easier to understand this concept when looking to how these values are calculated. First, a two-way contingency table of a word w and a class c is defined in the following way:

Table 2.1: General contingency table of a word w and a class c.

	class = c	**class $\neq$ c**	
contains w	A	B	A+B
$\neg$ contains w	C	D	C+D
	A+C	B+D	m=A+B+C+D

In Table 2.1, m is the number of documents, A is the number of documents in which word w and class c co-occur, B is the number of documents in which the word w occurs without class c, C is the number of documents in which class c occurs without word w and D is the number of documents in which neither class c nor word w occur [19].

Then, the Chi-square value between the word w and the class c can be computed as follows:

$$\chi^2(w,c) = \frac{m(AD-CB)^2}{(A+C)(B+D)(A+B)(C+D)} \underset{a}{\sim} \chi^2_{(1)} \tag{2.6}$$

This computation is repeated for all classes c_i, $i = 1, ..., L$ (L is the number of classes), and then the total Chi-square value of each word w (across all classes) is computed by an average mean, $\chi^2_{\text{avg}}(w) = \sum_{i=1}^{L} p(c_i)\chi^2(w,c_i)$, or by the maximum value, $\chi^2_{\text{max}}(w) = \underset{i=1,...,L}{max} \{\chi^2(w,c_i)\}$. Finally, the k words with the highest Chi-square values ($\chi^2_{\text{avg}}(w)$ or $\chi^2_{\text{max}}(w)$, depending on the researcher choice) are selected and the remaining words are discarded, from which results the reduced dimensions matrix X_k.

It should be noticed that in the Chi-Square the values are normalized and so the values of words can always be compared for the same class. However, a drawback of this normalization is that if a cell in the contingency table is low populated (e.g. when is related with a low frequency word) then it cannot be accurately compared with the χ^2 theoretical value (from the χ^2 distribution with one degree of freedom). Hence, the Chi-Square method does not work well for words with low frequency [19].

2.4.2.B Mutual information

Another feature selection method is the MI, that measures how much information the presence or absence of a specific variable, or word in the text data context, contributes to making the correct classification decision on a certain class. Hence, it quantifies the relevance of the feature set regarding the output (class) vector [20].

MI is based on another measure, called entropy, H, that represents the uncertainty of a random variable and is related to the probability of occurrence of an event. Thus, if each value of a discrete random variable has practically the same probability of occurrence, the entropy will be high, otherwise it will be low. The entropy of a given discrete random variable $\boldsymbol{X}$ with mass probability function defined by $p(x_i) = P(\boldsymbol{X} = x_i)$ with $x_i \in R_{\boldsymbol{X}}$, where $R_{\boldsymbol{X}}$ is the set of possible values of $\boldsymbol{X}$, is as follows:

$$H(\boldsymbol{X}) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{2.7}$$

However, it will be more useful the definition of joint entropy that, for two random variables $\boldsymbol{X}$ and $\boldsymbol{Y}$ with joint mass probability $p(x_i, y_j)$, is given by:

$$H(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{i=1}^{n} \sum_{j=1}^{L} p(x_i, y_j) \log p(x_i, y_j) \tag{2.8}$$

Thus, it can be proved that $I(\boldsymbol{X}, \boldsymbol{Y}) = H(\boldsymbol{X}, \boldsymbol{Y}) - H(\boldsymbol{X}) - H(\boldsymbol{Y})$. The definition of MI is:

$$I(\boldsymbol{X}, \boldsymbol{Y}) = -\sum_{i=1}^{n} \sum_{j=1}^{L} p(x_i, y_j) \log \left(\frac{p(x_i, y_j)}{p(x_i)p(y_j)} \right) \tag{2.9}$$

Note that MI will be zero when $\boldsymbol{X}$ and $\boldsymbol{Y}$ are statistically independent, since $p(x_i, y_j) = p(x_i)p(y_j)$.

When looking more specifically to a given word (from the set of features) and a given class c, the core part of the mutual information criterion (for which it will be used the same notation as for the MI) between the word w and the class c is given by $I(w, c) = p(w, c) \log \left(\frac{p(w,c)}{p(w)p(c)} \right)$. In fact, when considering the contingency table explained in Table 2.1, it is possible to estimate this value by:

$$I(w, c) = (A \times m) \log \left(\frac{A \times m}{(A + C)(A + B)} \right) \tag{2.10}$$

Then, this is repeated for all classes c_i, $i = 1, ..., L$, and then the global MI value of each word w (across all classes) is computed by an average mean, $I_{\text{avg}}(w) = \sum_{i=1}^{L} p(c_i) I(w, c_i)$, or by the maximum value, $I_{\text{max}}(w) = \underset{i=1,...,L}{max} \{I(w, c_i)\}$. Finally, just like with the Chi-Square feature selection method, the k words with the highest global MI values are selected and the remaining words are discarded, from which results the reduced dimensions matrix X_k.

Noticing that $I(w,c) = p(w,c)\log\left(\frac{p(w,c)}{p(w)p(c)}\right) = p(w,c)\left[\log(p(w|c)) - \log(p(w))\right]$, it is highlighted the fact that the score is strongly influenced by the marginal probabilities of words. Therefore, words with an equal conditional probability $p(w|c)$ will have a higher score if they are rare than if they are common words and so the MI scores are not comparable across words that have big differences in frequency for the same class [19]. This is not a problem with the Chi-Square method, as its values are normalized and so the values of words can always be compared for the same class, although this normalization also leads to the Chi-Square main drawback when any cell in the contingency table is low populated (already stated in the Subsection 2.4.2.A).

3

Text Categorization

Contents

In Chapter 3, the methods applied for the text categorization are introduced. First, a brief explanation on this concept is given, then, the different approaches (supervised, unsupervised, and semi-supervised learning) used to perform this task and some of their correspondent algorithms are described in detail. Finally, the performance evaluation metrics applied to those models are also introduced.

3.1 Text Categorization

Text categorization is basically the task of classifying documents into a defined set of categories (classes). This preferentially done through supervised learning but, since gathering sufficient labeled examples for training is not an easy and efficient task (manually labeling the hundreds or thousands of collected documents), it has been studied the possibility of doing it through unsupervised learning [41].

Thus, the next two Sections explain two fundamentally different types of approaches that can be used to perform the text categorization task: supervised learning, where pre-defined class labels are assigned to documents based on a training set of labeled documents, and unsupervised learning, where it is not necessary to have labeled documents for the process [40]. Then, a combination of both of these approaches is explored, semi-supervised learning.

First, it is introduced some notation that is used hereafter [42]. Let $X = (\boldsymbol{x}_1, ..., \boldsymbol{x}_m)$ be the set of m observations (usually assumed as i.i.d., independently and identically distributed, from a common distribution on $\mathcal{X}$), where $\boldsymbol{x}_i \in \mathcal{X}\ \forall i \in \{1, ..., m\}$. Another way of defining X is a $m \times n$ matrix, $X = (x_{ij}) \in \mathbb{R}^{m\times n}$, that contains the observations as rows and where n is the dimension of $\mathcal{X} \subset \mathbb{R}^n$.

Moreover, let $y_i \in \mathcal{Y}\ \forall i \in \{1, ..., m\}$ be the labels or targets of the observations $\mathbf{x}_i$, that can also be represented as a column vector, $\mathbf{y} = (y_i)^T \in \mathbb{R}^m$. In addition, assume there are L possible classes along vector $\mathbf{y}$.

3.2 Supervised Learning

Supervised learning is based on a training set, that is composed by pairs (x_i, y_i) (assumed to be realizations of i.i.d. random variables with support in $\mathcal{X} \times \mathcal{Y}$) with x_i an observation of the matrix X and y_i an element of the correspondent vector of labels $\mathbf{y}$. Hence, the goal is to train a function, based on the labeled data, able to assign a class to an unseen (and unlabeled) document.

These algorithms are divided into two categories: classification, that predicts exactly the class the observation belongs to, and regression, that predicts a numerical value based on previously observed data. In this work it is only mentioned the classification case. This being said, classification is a supervised learning task that consists in assigning labels (classes) to unlabeled observations [73]. In the particular case of text classification, the data instances are the documents or texts, that are characterized by words/tokens (the features of each data instance).

In order to perform such task it is used a model, the classifier and the classification process consists of two steps. First, the classifier uses a set of labeled observations and systematically learns from it through a learning algorithm (that defines the classifier), the induction step. Then, the trained classifier can be applied to a set of unlabeled observations and predict their labels, called the deduction step.

Typically, the dataset is divided into two subsets: training set and test set. The training set is the one used in the induction phase and the test set is the one used to evaluate the classifiers performance on unseen data. As a means to obtain a classifier with a good generalization performance, that is, a model that correctly predicts class labels of observations that were never seen before, these data subsets should be independent between them [73]. This will be covered in more detail in Section 3.5.

Moreover, a multi-class classification problem, as the name suggests, is a classification problem with more than two possible classes (labels) for each observation of the dataset. It is different from multi-label classification problems, where each observation can be classified as multiple labels (the labels are not mutually exclusive), since each observation is classified into only one of the L possible classes.

3.2.1 Naive Bayes

The NB classifier is one of the most popular between all classifiers due to its simplicity. The main goal of this model is to assign an a posteriori class probability to each observation, $p\left(y_i|x_i\right)$, and from there obtain the classification results [32]. For simplicity, the index i in $x_i = \left(x_{i1}, x_{i2}, ..., x_{in}\right)$ will be omitted and hence the document x_i will be represented as $\boldsymbol{x} = \left(x_1, x_2, ..., x_n\right)$. Therefore, the aim is to compute $p\left(y_i|\boldsymbol{x}\right)$, for each $y_i \in \mathbf{y}$ and then assign to the observation $\boldsymbol{x}$ the class y_i with the highest posterior class probability. From the Bayes' theorem it comes:

$$p\left(y_i|\boldsymbol{x}\right) = \frac{p\left(\boldsymbol{x}|y_i\right)p\left(y_i\right)}{p\left(\boldsymbol{x}\right)} \tag{3.1}$$

Beginning with the numerator of the above expression, (3.1), it should be recalled that $p\left(\boldsymbol{x}|y_i\right)p\left(y_i\right) = p\left(\boldsymbol{x}, y_i\right) = p\left(x_1, x_2, ..., x_n, y_i\right)$. Moreover, using the basic rule $p(A,B) = p(A|B)p(B)$, it is obtained that:

$$\begin{aligned} p\left(x_1, x_2, ..., x_n, y_i\right) &= p\left(x_1|x_2, ..., x_n, y_i\right)p\left(x_2, ..., x_n, y_i\right) && (3.2)\\ &= p\left(x_1|x_2, ..., x_n, y_i\right)p\left(x_2|x_3, ..., x_n, y_i\right)...p\left(x_n|y_i\right)p(y_i) && (3.3)\end{aligned}$$

The NB assumes that the features are conditionally independent given the class, that is:

$$p\left(x_j|x_1, ..., x_{j-1}, x_{j+1}, ..., x_n, y_i\right) = p\left(x_j|y_i\right) \tag{3.4}$$

or, written in an equivalent way, $p\left(x_1, ..., x_n|y_i\right) = \prod_{j=1}^{n} p\left(x_j|y_i\right)$. This is known as the conditional independence assumption. Hence, it follows that:

$$p\left(\boldsymbol{x}|y_i\right) p\left(y_i\right) = \prod_{k=1}^{n} p\left(x_k|y_i\right) p\left(y_i\right) \tag{3.5}$$

Going back to the expression (3.1), since $p(\boldsymbol{x})$ is constant given the input observation, that is, it does not depend on the class, and given the assumption (3.5), the next relation is true:

$$p\left(y_i|\boldsymbol{x}\right) \propto \prod_{k=1}^{n} p\left(x_k|y_i\right) p\left(y_i\right) \tag{3.6}$$

Note that this holds since it is supposed to be assigned one and only one class to each observation.

Finally, using the Maximum A Posteriori (MAP) estimation, the observation $\boldsymbol{x}$ is classified as the class $\hat{y}$, that is:

$$\hat{y} = \underset{y_i}{argmax} \ \ p\left(y_i|\boldsymbol{x}\right) = \underset{y_i}{argmax} \prod_{k=1}^{n} p\left(x_k|y_i\right) p\left(y_i\right) \tag{3.7}$$

In order to compute the prior estimates $\hat{\theta}_{x_k|y_i} = p(x_k|y_i, \theta)$ and $\hat{\theta}_{y_i} = p(y_i|\theta)$, it is solved the system of partial derivatives of $log\left(p(\theta|\boldsymbol{x}, \mathbf{y})\right)$, using Lagrange multipliers that will impose the constraint of the sum of the word probabilities in a class being equal to one (this is explained in more detail with the algorithm presented in Subsection 3.4.3).

It should be noticed that the naive assumption used in (3.5) is a quite strong assumption and in the majority of cases is not satisfied, which is the worst drawback of the NB classifier. However, even when this condition is violated, this classifier performs impressively well [33]. On another hand, despite being a good classifier, NB does not work so well as an estimator of $p\left(y_i|\boldsymbol{x}\right)$ since, as it assumes the features are conditionally independent, the probabilities will be incorrect if this naive assumption is not satisfied.

3.3 Unsupervised Learning

In the unsupervised learning, the vector of labels, $\mathbf{y}$, is not known and the goal is to find an interesting structure in the data, that is, new patterns in X [42]. This can be performed by clustering the data into several categories based on its features. Clustering is a technique used when analyzing data, dividing it in different subsets, called clusters, and rearranging the observations into those groups based on similarity or dissimilarity, providing subsets of the dataset with similar data in each of them [38]. The set of clusters is not known a priori. Therefore, using clustering algorithms, the dataset X can be described as $\{\boldsymbol{x}_1, ..., \boldsymbol{x}_n\} = \bigcup_{i=1}^{k} C_k$, where $C = \{C_1, ..., C_k\}$ is the clustering structure and $C_i \bigcap C_j = \emptyset$ for $i \neq j$, which means that each observation belongs to one and only one cluster [74].

In order to measure the similarity or dissimilarity between two observations, it can be used distance measures or dissimilarity measures. On the one hand, a distance measure should verify the usual distance properties (symmetry, identity of indiscernibles and triangle inequality) and achieves its minimum value, zero, when comparing two identical observations. Minkowski metric, Jaccard coefficient and Euclidean distance are some possible examples for distance measures. Alternatively, a similarity measure should also have the symmetry property but when comparing two identical observations it obtains its largest value. Some examples of similarity measures are Pearson Correlation Measure, Extended Jaccard Measure, Dice Coefficient Measure and Cosine Measure. The last one is usually the chosen one when comparing text data (as suggested in [36]).

The cosine similarity measure between two words x_i and x_j is given by $s(x_i, x_j) = \frac{x_i^T \cdot x_j}{\|x_i\| \cdot \|x_j\|}$. Since each word is represented by a vector with non negative values (the weights of the word in each document), then the cosine similarity value will also be non-negative and from its expression it is straightforward that its value is bounded between 0 and 1 [36]. Note that the cosine similarity is independent of document length. Moreover, the cosine similarity measure can be transformed as a dissimilarity measure (and vice versa) if it is applied the transformation $d(x_i, x_j) = 1 - s(x_i, x_j)$, due to its bounding values.

3.3.1 K-Means

The K-means clustering algorithm is one of the distance-based partitioning algorithms. Essentially, the algorithm starts by choosing k initial centroids, that is, the set of central points around which the clusters are built and that usually belong to the dataset (but not necessarily). Then, each point from the dataset is assign to the cluster with the nearest centroid, according to some distance/dissimilarity (e.g. Euclidean, Manhattan, Cosine, [37]). After that, new centroids are computed, by taking the mean value of all the points in that cluster. The last two steps are repeated until convergence, that can be interpreted as the centroids not moving.

The cluster centers are computed as the mean of all points in the cluster, that is, $\mu_k = \frac{1}{m_k} \sum_{q=1}^{m_k} x_q$, with m_k being the number of observations that belongs to cluster k [74]. Thus, the K-means algorithm is described in Algorithm 3.1.

Note that one possible way to choose the initial centroids or cluster centers is to assign observations randomly to each cluster. The number of clusters, k, is pre-defined. This leads to the main disadvantage of the K-means algorithm, that is the sensitivity to initial positions of the centroids, since it can influence in a great scale the quality of the clusters partition [34]. Also, choosing the value k is not an easy task and even impractical, requiring greater knowledge in the clustering field [38]. However, this algorithm is frequently chosen, as it is very easy to implement and computationally faster than the others [39].

Algorithm 3.1: K-means Algorithm

```
Input   : $X$, $k$, $dist$.
Set $k$ initial centers, $\mu_i$ for $i = 1, ..., k$.
while convergence is not reached do
    Assign observations of $X$ to the closest cluster center, based on $dist$;
    Update the cluster centers $\mu_i$ for $i = 1, ..., k$.
Output: $k$ clusters.
```

3.3.2 DBSCAN

DBSCAN, introduced by [35], was the first density based clustering algorithm, that is, identifies the clusters based on the idea that within each cluster the density of points is significantly higher than outside of the cluster. This algorithm takes into account essentially two parameters: *Eps* and *MinPts*. *Eps*, or ϵ, consists of a radius (of an arbitrary distance measure) that represents the minimum distance below which two points are considered close to each other, being the ϵ-neighborhood of a point p defined by $N_\epsilon(p) = \{q \in X : \text{dist}(p, q) \leq \epsilon\}$, where X is the dataset of points [35, 75]. In addition, *MinPts* is the minimum number of points that need to be in the ϵ-neighborhood of the point p.

There are three kinds of points considered by this algorithm. The core points are defined as the points that have a ϵ-neighborhood with at least *MinPts*. The border (or density-reachable) points are the points that can be directly reached from a core point, that is, have a distance smaller than ϵ from a core point but its ϵ-neighborhood has less than the *MinPts*. At last, the noise is the set of all points that are not directly reached by a core point and so do not belong to any cluster. Hence, a cluster consists of a set of core points, that are close to each other (according to some distance measure), and a set of border (non-core) points that are close to one (or more) of those core points. Moreover, higher values of *MinPts* or lower values of ϵ refer to higher density necessary to form a cluster. As it is described in [76], the original algorithm of DBSCAN is described in Algorithm 3.2.

In this description there are some details that need to be explained. The general idea is that all the points start with no cluster associated (undefined) and then the algorithm iterates over each point p of the dataset X in order to build the clusters and label each point to the correct cluster. Note that the points that are already labeled (not undefined) are skipped. It is used an auxiliary function, *RangeQuery*, that takes the dataset X, the radius ϵ, the point p and a distance function *dist*, and gives the set of neighbors of the point p, that is, the ϵ-neighborhood of p. With this information, it is possible to verify if p is a core point. If it is not, p is labeled as *noise* (for now) and the algorithm picks the next point of the dataset, whereas, if p is a core point, then a new cluster c is formed by p and its neighbors. In the latter case, *RangeQuery* function needs to be applied to each q point in the ϵ-neighborhood of p to verify if the neighbor points themselves are also core points. If q is not a core point, then the algorithm moves on to the next point in the ϵ-neighborhood of p. Otherwise, add its neighbors to the cluster. Thus, the set

Algorithm 3.2: DBSCAN Algorithm

```
Input   : $X$, $\epsilon, MinPts, dist.$
Set $label$ of all points in $X$ as $undefined$.
for $p$ in $X$ do
    if $label(p) \neq undefined$ then
        continue
    $N_p \longleftarrow RangeQuery(X, \epsilon, p, dist)$
    if $|N_p| < MinPts$ then
        $label(p) \longleftarrow Noise$
        continue
    $c \longleftarrow$ next cluster label
    $label(p) \longleftarrow c$
    $Q \longleftarrow N_p \setminus \{p\}$
    for $q$ in $Q$ do
        if $label(q) = Noise$ then
            $label(q) \longleftarrow c$
        if $label(q) \neq undefined$ then
            continue
        $N_q \longleftarrow RangeQuery(X, \epsilon, q, dist)$
        $label(q) \longleftarrow c$
        if $|N_q| < MinPts$ then
            continue
        $Q \longleftarrow Q \cup N_q$
```

of points in Q (i.e. the points in the neighborhood of p except itself) and the core point p will define the cluster c.

Therefore, this algorithm works quite well in the presence of outliers and at detecting them and also it can find arbitrarily shaped clusters (in contrast to K-means). On the other hand, DBSCAN is very sensitive to its parameters (*MinPts* and ϵ) and, when the cluster density varies or the dataset is too sparse, it fails to identify the clusters [39].

3.4 Semi-Supervised Learning

SSL is a combination of both supervised and unsupervised learning. It is the most suitable approach when the majority of observations from the dataset are unlabeled but there are a few observations with labels associated, since it can combine the information of both parts and learn the underlying distribution of the data [42]. In the "standard" semi-supervised learning setting (the one that will be used throughout this dissertation), the dataset X is divided into two parts:

X_l $:= (\boldsymbol{x}_1, ..., \boldsymbol{x}_l)$, associated with labels $\mathbf{y}_l := (y_1, ..., y_l)$

X_u $:= (\boldsymbol{x}_{l+1}, ..., \boldsymbol{x}_{l+u})$, without labels associated

where l and u are the number of labeled and unlabeled observations, respectively, and obviously $l + u =$

m. Here, SSL works like supervised learning but with additional information on the distribution of the observations (the unlabeled data), having the same goal of predicting a label for each observation.

Nevertheless, there are other forms of semi-supervised learning, for instance semi-supervised learning with constraints. In this case, SSL works like unsupervised learning but with some constraints, for example knowing beforehand that some observations have (or do not have) the same label.

Moreover, SSL can be classified into four main kinds of approaches, namely, Self-labeled Methods, Generative Methods, Low-Density Separation Methods, Graph-Based Methods. The algorithms resulting from each of these approaches are based in a set of assumptions.

In the next Subsections it will be discussed the set of assumptions on which SSL relies as well as some of the algorithms from each of the above mentioned approaches, how they work and the main ideas behind them.

3.4.1 Semi-Supervised Learning Assumptions

At first sight, there is a question that immediately arises: is the additional (unlabeled) data significant and does it really help to improve classification performance? Usually the answer is yes. However, the distribution of observations (including the unlabeled ones) must be relevant for the classification problem or, in other words, the knowledge on $p(\boldsymbol{x})$, obtained through the unlabeled observations, has to carry useful information for the inference of $p(\mathbf{y}|\boldsymbol{x})$. Otherwise, SSL will not lead to an improvement over supervised learning and it could even worsen the results by misguiding the inference.

Hence, in order to get SSL to work, there are three assumptions that must hold:

Smoothness assumption: If two points x_1, x_2 in a high-density region are close, then so should be the corresponding outputs y_1, y_2.
This basically means that the label function is smoother in high-density regions than in low-density regions, that is, if two observations are linked by a path of high density (e.g., if they belong to the same cluster), then their outputs (classes) are likely to be close (i.e. similar), however, if they are separated by a low-density region, then their outputs are not necessarily close.

Cluster assumption: If two observations are in the same cluster, they probably belong to the same class.
This means that, in general, observations that belong to distinct classes do not appear in the same cluster, but it does not say that each class forms a single compact cluster.
Moreover, looking at clusters as high-density regions, it is possible to define an equivalent assumption: Low density separation assumption. It says that the decision boundary should lie in a low-density region (otherwise it might cut a cluster into two different classes).

Manifold assumption: The (high-dimensional) data lies (roughly) on a low-dimensional manifold. This is an important assumption in a sense that it prevents the so called curse of dimensionality (already stated in Section 2.4), since in this case (if the data lies indeed on a low-dimensional manifold) the classification algorithm can function in a lower dimensional space (the manifold).

3.4.2 Self-labeled Methods

The Self-labeled methods are usually based on supervised classification algorithms and the idea behind them is to classify the unlabeled observations from the dataset according to their most confident predictions [43]. The simplest and probably the first idea of SSL is Self-learning, also known as self-training, self-labeling or decision-directed learning and consists in repeatedly applying a supervised learning method to the dataset. It starts by training the model with the labeled data and then, with that model, the labels of some unlabeled observations are predicted using the current decision function. Afterwards, the model is retrained with the labeled data and the new labeled observations. This process is repeated until all the unlabeled observations are associated with some label.

Even though it is a very simple algorithm, it has a notorious issue that relates with the fact that misclassified observations will be spread to the next classification iterations, since the model assumes that its own predictions tend to be correct [45].

There are other variations of the standard Self-learning algorithm, for example, Co-training, that uses multiple classifiers (instead of just one) and trains them on different and preferably disjoint sets of features, combining their predictions in order to decrease the classification error.

3.4.3 Generative Methods

When we talk about classification, there are usually two main approaches: generative or discriminative. The generative models include the distribution of the data itself and can tell us how likely a given observation is. They intend to capture the class-conditional density $p(\boldsymbol{x}|\mathbf{y})$ (in contrast to discriminative models that estimate the conditional probability $p(\mathbf{y}|\boldsymbol{x})$) using some unsupervised learning procedure and then, by the Bayes theorem, it is possible to deduce a predictive density, using:

$$p(\mathbf{y}|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathbf{y})p(\mathbf{y})}{\sum_y p(\boldsymbol{x}|y)p(y)} \tag{3.8}$$

Moreover, since $p(\boldsymbol{x}|\mathbf{y})p(\mathbf{y}) = p(\boldsymbol{x}, \mathbf{y})$ (joint probability), new data observations (x_i, y_i) can be generated from this joint density of the data. In this way, there are essentially two possible interpretations of semi-supervised learning when using a generative model. The first one is interpreted as classification (supervised learning) with additional information on the marginal density, $p(\boldsymbol{x})$, and the other is interpreted as clustering (unsupervised learning) with additional information. In the last case, this additional information generally corresponds to the true labels of a subset of observations from the dataset, however it can also be defined by more general constraints.

An interesting aspect of the generative methods is that any knowledge on the data or the structure of the problem can be easily integrated by modeling it, being this one of the advantages of this kind of approach.

3.4.3.A Expectation-Maximization Algorithm

The generative model presented below rests on three essential assumptions, that are also used by the NB classifier:

1. Data (documents) is produced by a mixture model;
2. Mixture components and classes have a one-to-one correspondence between them;
3. Mixture components (classes) are multinomial distributions of individual words.

Based on these assumptions, and assuming a vocabulary of size $|\mathcal{X}|$ and that each document, x_i, has exactly $|x_i|$ words in it, it is possible to generate a document using this kind of model.

In an abstract way, in order to create a new document, firstly a biased L-sided dice should be rolled to set the document's class and then another biased dice, $|\mathcal{X}|$-sided, corresponding to that class is picked up. Next the last dice is rolled $|x_i|$ times and the number of times each word occurs is counted. With these word counts it is obtained the generated document.

Now, in a more formal way, it can be said that the new document is generated according to a probability distribution defined by the parameters for the mixture model, denoted θ. This probability distribution is then described by a mixture of components c_j, with $j \in \{1, ..., L\}$. Therefore, in order to generate the document x_i, we should start by selecting a mixture component based on the mixture weights (or class probabilities) $p(c_j|\theta)$ and then use it to generate a document according to the parameters of this mixture component, with distribution $p(x_i|c_j;\theta)$. Thus the likelihood of seeing document x_i is obtained by computing a sum of total probability over all mixture components, that is:

$$p(x_i|\theta) = \sum_{j \in \{1,...,L\}} p(c_j|\theta) p(x_i|c_j;\theta) \tag{3.9}$$

It should be noticed that each document x_i has a class label associated, y_i, and by the assumptions already stated, if c_j represents a certain component of the mixture model that has generated document x_i then it holds $y_i = c_j$.

The goal remains the same, that is, compute $p\left(y_i|\boldsymbol{x}\right)$ (in the previous notation, $p(c_j|x_i;\theta)$), for each $y_i \in \mathbf{y}$ and then assign to the observation $\boldsymbol{x}$ the class with the highest posterior class probability, $\hat{y}$.

However, the formula above is only applicable to labeled data and not unlabeled data. One possible solution is to use the EM algorithm, in which MAP parameter estimates are locally found for the generative model.

The EM algorithm consists in a set of steps that may involve both labeled and unlabeled data in the correspondent parts. The first step is to define a NB (or other) classifier as in the supervised manner (Subsection 3.2.1), using only the labeled training data. Then, with this classifier, the unlabeled data is classified but instead of trying to find the most likely class it is calculated the probabilities associated with each class. These last estimated class probabilities are used as the true class labels for its observations and a new NB model is build. Until the algorithm converges to a stable classifier and all the data is labeled, the procedure of classifying the unlabeled data (**E** step) and rebuilding the classifier (**M** step) is iterated, as described in Algorithm 3.3.

Algorithm 3.3: EM algorithm for Semi-Supervised Text Classification

```
Input: $X_l$ (labeled documents) and $X_u$ (unlabeled documents).
  • Build a NB classifier, $\hat{\theta}$, from $X_l$.
  • While the classifier parameters improve (measured by the change in $l(\theta|\boldsymbol{x}, \mathbf{y})$):
      1. Compute $p(c_j|x_i; \hat{\theta})$ for each observation in $X_u$;
      2. Re-build the classifier $\hat{\theta}$ using the estimated probabilities from step 1. and the MAP
         estimation.
Output: Classifier, $\hat{\theta}$, that predicts a class label for each input unlabeled document.
```

It should be noticed that the expected log probability measured along the Algorithm 3.3 is defined as follows:

$$\begin{aligned} l(\theta|\boldsymbol{x}, \mathbf{y}) &= log(p(\theta|\boldsymbol{x}, \mathbf{y})) = log(p(\theta)p(\boldsymbol{x}, \mathbf{y}|\theta)) \\ &= log(p(\theta)) + log(p(\boldsymbol{x}, \mathbf{y}|\theta)) \\ &= log(p(\theta)) + log(\prod_{x_i \in X} p(x_i|\theta)) \\ &= log(p(\theta)) + \sum_{x_i \in X_u} log\left(\sum_{j \in \{1,\ldots,L\}} p(c_j|\theta)p(x_i|c_j; \theta)\right) \\ &+ \sum_{x_i \in X_l} log(p(y_i = c_j|\theta)p(x_i|y_i = c_j; \theta)) \end{aligned} \tag{3.10}$$

Here it is used the relation in Equation (3.9) but the cases of each set, X_l and X_u, are separated. This is due to the fact that with the labeled data, X_l, since the label y_i associated with the generating component is given, it is not necessary to go through all mixture components (just the one corresponding to y_i), as it is in the case of the unlabeled data, X_u. Also, with the same justification as explained in Subsection 3.2.1, the constant terms are not considered here.

A problem that immediately stands out is the log of sums that appears in the part of the unlabeled data. This leads to computational intractability when trying to maximize by partial derivatives. One possible solution is an iterative hill-climbing approach to finding a local maxima of the parameter [44]. In this case, when measuring the change in $l(\theta|\boldsymbol{x}, \mathbf{y})$ in the algorithm, instead of searching for no change the aim is to get a below-threshold change in it. This will correspond to the height of the surface on which EM is hill-climbing. Another drawback is that the algorithm relies on the mixture model assumptions and if they are not satisfied then the unlabeled data may worsen the results.

3.4.4 Low-Density Separation Methods

It can be inferred from the name that these methods try to directly implement the low-density separation assumption, that is, they try to push the decision boundary away from the unlabeled observations, assuming that classes are well separated. Typically, to do so, it is used a maximum margin algorithm, for instance, Support Vector Machines. When the goal is to maximize the margin for unlabeled and labeled observations (and not just the labeled ones), this method is called the Transductive Support Vector Machines [42]. In addition, it is clear that this method implements the idea of transductive learning, since it includes test observations in the computation of the margin. This will not be further discussed as it will not be applied in this work.

3.4.5 Graph-Based Methods

In the graph-based methods, as the name suggests, the data is represented by a graph, where the nodes are the data observations (both labeled and unlabeled) and the edges represent the pairwise distances or similarities between them. In this graph, a missing edge corresponds to an infinite distance or zero similarity (when one of the nodes does not have an associated label) and in order to avoid these missing edges the information from the labeled nodes propagates through the graph until all nodes are labeled.

Thus, one can say that these kind of algorithms rely on the geometry of the data induced by both labeled and unlabeled observations (improving the supervised methods that use only the labeled observations). This geometry is then represented by a graph $g = (V, E)$ where V is the set of nodes (training observations) and E is the set of real edge weights given by a function $w : E \rightarrow \mathbb{R}$. The values $w(e)$ of each edge, e, can be represented in a weighted adjacency matrix (or just weight matrix) W that is defined by:

$$W_{ij} := \begin{cases} w(e) & \text{if } e = (i,j) \in E \\ 0 & \text{if } e = (i,j) \notin E \end{cases} \tag{3.11}$$

In fact, W_{ij} is usually given by a symmetric positive function W_X that depends on the data set $X = (x_1, ..., x_m)$ and it follows that $W_{ij} = W_X(x_i, x_j) \geq 0$ and $W_X > 0$ iff x_i and x_j are ”neighbours”.

Another important matrix that should be computed is the diagonal matrix D, defined by $D_{ii} := \sum_j W_{ij}$, and also called the degree matrix of the graph g. This is important since most of these graph-based methods refer to the graph g through the graph Laplacian, that can be defined in a variety of ways, being the main two the following:

Normalized graph Laplacian: $\mathcal{L} := \mathrm{I} - \mathrm{D}^{-\frac{1}{2}}\mathrm{W}\mathrm{D}^{-\frac{1}{2}}$

Unnormalized graph Laplacian: $L := \mathrm{D} - \mathrm{W}$

Note that, if the distance of two points is obtained by minimizing the set of path distances over all paths connecting those points, this can be interpreted as an approximation of the geodesic distance of the two points with respect to the manifold of observations. For this reason, it might be said that graph-based methods are built on the manifold assumption.

Moreover, since these methods predict labels for the unlabeled nodes, that is, they return only the values of the decision function on the unlabeled observations (instead of the all decision function), they are intrinsically transductive methods. Therefore, the main issue with graph-based methods is that their performance strongly depends on the graph structure, that should fit the classification goal in order to perform well, and edge weights [45].

3.4.5.A Label Propagation Algorithm

In the graph-based methods for semi-supervised learning, one of the most popular techniques is the Label Propagation (LP). The general idea is to propagate labels on the graph g, starting with the labeled nodes, X_l, and the correspondent set of labels, Y_l. In this way, each node starts to propagate its label to its neighbors, through the edges, and this procedure is iterated until convergence.

As suggest by [77], cited by [78], there is an algorithm where the estimated labels are defined as $\hat{Y} = (\hat{Y}_l, \hat{Y}_u)$. In this case, $\hat{Y}_l$ should be the same as Y_l, but there are other algorithms where they may differ from each other.

The affinity matrix or weight matrix for this algorithm is defined as the Gaussian kernel of width σ, that is, $W_{ij} = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$. Another matrix that should be defined is the probabilistic transition matrix $T = D^{-1}W$, where T_{ji} represents the probability of going from node j to node i. This values are calculated in the following way:

$$T_{ji} = p(j \to i) = \frac{w_{ij}}{\sum_{k=1}^{m} w_{kj}} \tag{3.12}$$

In order to facilitate the understanding, assume from now on that there are only two possible classes,

1 and 0. The unknown labels will be set as -1. Thus, the procedure is described in Algorithm 3.4.

Algorithm 3.4: LP algorithm

```
• Compute the weight matrix $W$.
• Compute diagonal matrix $D$ by $D_{ii} \longleftarrow \sum_j W_{ij}$ and the probabilistic transition matrix $T = D^{-1}W$.
• Initialize $\hat{Y}^{(0)} = (y_1, ..., y_l, -1, ..., -1)$.
• Until convergence to $\hat{Y}^{(\infty)}$, iterate:
    1. $\hat{Y}^{(t+1)} \longleftarrow T\hat{Y}^{(t)}$;
    2. $\hat{Y}_l^{(t+1)} \longleftarrow Y_l$.
• Label $x_i$ by $\hat{y}_i^{(\infty)}$.
```

3.4.5.B Label Spreading Algorithm

According to [79], there is another similar algorithm, the Label Spreading (LS) . The main idea of this algorithm is that, at each iteration, the nodes should use both information from its neighbors and from its initial label information. As in the LP algorithm, one should start by defining the affinity matrix, W, with the slight difference that now each element of the diagonal of W is set to zero in order to avoid self-reinforcement. Then, the matrix W is normalized symmetrically, for the convergence of the next iteration.

The next step is a loop, where at each iteration each node collects information from its neighbors (that will correspond to the first term of the equation to solve) and, at the same time, preserves its initial information (that corresponds to the second term). Here, the parameter α corresponds to the relative amount of information that is obtained from the neighbors and from the initial information $(1 - \alpha)$.

The last step of the algorithm consists in labeling the unlabeled observations. The class of which an observation has received most information through the whole iteration process is the chosen label for it. Hence, the procedure is defined in Algorithm 3.5.

Algorithm 3.5: LS algorithm

```
• Compute weight matrix $W$ with $W_{ii} \longleftarrow 0$.
• Compute diagonal matrix $D$ by $D_{ii} \longleftarrow \sum_j W_{ij}$ and the symmetric matrix $S = D^{-\frac{1}{2}}WD^{-\frac{1}{2}}$.
• Initialize $\hat{Y}^{(0)} = (y_1, ..., y_l, -1, ..., -1)$ and define the parameter $\alpha \in [0, 1)$.
• Until convergence to $\hat{Y}^{(\infty)}$, iterate $\hat{Y}^{(t+1)} \longleftarrow \alpha S\hat{Y}^{(t)} + (1 - \alpha)\hat{Y}^{(0)}$.
• Label $x_i$ by $\hat{y}_i^{(\infty)}$.
```

3.5 Evaluation Metrics

After classifying data, there is the need to understand if the data was well classified and if the classifier can accurately predict the class labels of new observations (that were not used to train the classifier) or, in other words, if the classifier has good generalization performance [73]. Therefore, an evaluation metric is essentially a measure of the classifier performance and effectiveness, where different metrics evaluate different characteristics of the classifier [48].

Besides measuring the generalization ability of classifiers, the evaluation metrics can also be utilized to determine which classifier has better performance when comparing a set of different classifiers or even to select the best solution produced by a classifier when varying some of its parameters [48]. As already introduced in Section 3.2, the dataset is divided into two subsets, training set and test set, being the test set the one used to evaluate the classifiers performance.

3.5.1 Generalization Performance

In this case, the evaluation metrics are used with the aim of measuring and resuming the quality of the trained classifier when tested with unseen data [48]. Note that this is primarily used in supervised learning.

The effectiveness of a classifier can then be obtained by comparing the actual labels of the observations with the labels predicted by the classifier [73]. The confusion matrix is a table that is usually used to organize this information and to better understand and interpret the results. It is similar to the contingency table described in Table 2.1 and, for a binary classification problem, it can be defined as follows:

Table 3.1: Confusion matrix for binary classification

		Predicted Class	
		Class = 1	**Class = 0**
Actual Class	**Class = 1**	TP	FN
	Class = 0	FP	TN

where TP (true positives) correspond to the number of observations correctly classified as class 1 (positive), TN (true negatives) correspond to the number of observations correctly classified as class 0 (negative), FP (false positives) correspond to the number of observations misclassified as class 1 and FN (false negatives) correspond to the number of observations misclassified as class 0.

From these values, there can be derived various evaluation metrics, summarizing this information into a single value. The most common one is the accuracy or overall accuracy that measures the proportion of correctly predicted observations over the test set and can be computed as

$$accuracy = \frac{\text{Nr of correct predictions}}{\text{Total nr of predictions}} = \frac{TP + TN}{TP + FP + FN + TN} \tag{3.13}$$

There are also the Sensitivity (or Recall of class 1), Specificity (or Recall of class 0) and Precision per class metrics. Sensitivity is the rate of observations correctly predicted as positives (class 1) among all observations that are actually positives, specificity is the rate of observations correctly predicted as negatives (class 0) among all observations that are actually negatives and precision is the rate of observations correctly predicted as positives among all observations that were predicted as positives. Hence, they can be defined in the following way:

$$Sensitivity = Recall(1) = \frac{TP}{TP + FN} \tag{3.14}$$

$$Specificity = Recall(0) = \frac{TN}{TN + FP} \tag{3.15}$$

$$Precision = \frac{TP}{TP + FP} \tag{3.16}$$

These metrics can be combined into other metrics such as the F_β-measure (or F_β-score), that combines both precision (3.16) and recall (3.14) and is given by:

$$F_\beta - score = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times Precision + Recall} \tag{3.17}$$

where β is a coefficient that determines whether to give more weight to precision or to recall, that is, a β value smaller than 1 gives more importance to precision and a β value higher than 1 gives more importance to recall [49]. This is useful since it could be the case where it is more important to minimize FP (β value smaller than 1) or the case where it is more important to minimize FN (β value higher than 1).

When $\beta = 1$, both precision and recall have the same importance and the evaluation metric is called F_1-measure (or F_1-score). F_1-score is the harmonic mean between precision an recall and is defined by the equation (3.17) with $\beta = 1$:

$$F_1 - score = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.18}$$

Although these metrics are very effective with binary classification problems, some adjustments need to be done in order to assess the performance of multi-class classification problems and only a few of the previous metrics can be adapted to that.

Starting with the confusion matrix, in a multi-class classification problem it will be defined by a $L \times L$ matrix, where L is the number of classes and the entries $CM_{i,j}$ correspond to the number of observations with class i that were predicted as class j [55]. Therefore, each class i has its own TP_i, TN_i, FP_i and FN_i values, defined in a similar way as in the binary case but with an one-vs-rest comparing logic.

When looking at multi-class evaluation metrics, the concept of true or false positives and true or false negatives slightly changes, since they must be considered under the context of a particular class. In that way, for a given class i, the true positives value, TP_i, corresponds to the value in the main diagonal of the confusion matrix, $CM_{i,i}$; the false positives value, FP_i, corresponds to the sum of all values in the column of class i except the diagonal value, $\sum_{\substack{r=1 \\ r\neq i}}^{L} CM_{r,i}$; the false negatives value, FN_i, corresponds to the sum of all values in the row of class i except the diagonal value, $\sum_{\substack{c=1 \\ c\neq i}}^{L} CM_{i,c}$; the true negatives value, TN_i, corresponds to the sum of all the values of the confusion matrix excluding the values of the row and column of class i, $\sum_{\substack{r=1 \\ r\neq i}}^{L} \sum_{\substack{c=1 \\ c\neq i}}^{L} CM_{r,c}$.

Afterwards, a general value can be obtained combining these values from each class. One way of doing it is to compute a macro average, M, that is, averaging the evaluation metric results over all classes. Another way is to use the grouped results (cumulative values) and it is called micro average, μ. However, when including all the classes, which is usually the case, this will match the accuracy. Whilst macro-averaging treats all classes equally, micro-averaging favors bigger classes [47].

Thus, the macro-averaged F_β-score, F_{β_M}, is defined as follows:

$$F_{\beta_M} = \frac{1}{L} \sum_{i=1}^{L} F_{\beta_i} \tag{3.19}$$

Accuracy, on the other hand, when computed for a multi-class classifier is given by the fraction of correct predictions over the total number of observations, m, as follows:

$$accuracy = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}_{(\hat{y}_i = y_i)} \tag{3.20}$$

where $\mathbb{1}$ is the characteristic function, $\hat{y}_i$ is the predicted value of the $i-$th observation, and y_i is the corresponding true value.

3.5.2 Model Selection

The classifier has some model parameters that determine the model complexity. Although, in general, a high complexity leads to better discriminating ability it also comes with a risk of overfitting, that is what happens when the classifier has a extremely good performance when predicting the observations from the training set but completely fails when applied to new, unseen data (it has a bad generalization performance) [52]. On the other hand, there is also the risk of underfitting when with lower complexity, which means that the model is not able to capture at all the relation between the variables [53]. Therefore, the aim is to find the set of the classifier parameters that reflect the right balance between those two issues.

In order to assess that, the evaluation metrics can be also applied as evaluators for model selection, having the task of determining the best classifier among all the possibilities [48]. In this case, the training set needs to be splitted into a smaller training set and a validation set [52, 53]. The validation set is used as the test set but with the goal of selecting the best model parameters based on the validation error, that should be as low as possible.

There are several techniques that can be applied to perform this model selection, being k-fold cross-validation one of the most popular ones. This technique consists of splitting the training set into k different subsets, called k-folds, in a way that there is no overlapping between these k subsets. Then, the model is trained in a set of $k-1$ folds (that represent the training set) and thereafter it predicts the observations of the remaining subset of the k folds (that will work has the validation set) and its performance is evaluated and stored. These steps (training, predicting and evaluating the performance) are repeated k times and in each time a different fold of the k folds is utilized as the validation set [52, 53]. Finally, the stored values of the measured performance are averaged and that is the cross-validated performance. This entire process should be repeated for different combinations of parameters so that the best model parameters can be obtained, that is, the parameters that lead to the model with the best cross-validated performance. Note that this technique is usually referred as k-fold CV and, even more so, if the value k coincides with the total number observations in the training set, it is a special case of cross-validation, called leave-one-out method [52].

3.5.3 Imbalanced Data Sets

When dealing with imbalanced data sets, using standard evaluation metrics is most likely not the best approach for performance evaluation, as they are usually biased and tend to give more importance to the majority classes rather than to the minority ones (see Subsection 2.2.5).

Although accuracy is easy to compute and with low complexity, suitable for multi-class problems and easy to interpret, it assigns more weight to the majority classes than the most rare ones, which turns it into a misleading metric since the classifier will not perform well on the minority classes [54]. However, there exists an alternative to the standard accuracy, called balanced accuracy, given by [80]:

$$balanced\ accuracy = \frac{1}{\sum_{i=1}^{m} \hat{w}_i} \sum_{i=1}^{m} \mathbb{1}_{(\hat{y}_i = y_i)} \hat{w}_i \tag{3.21}$$

where $\hat{w}_i$ is an adjusted observation weight, based on the inverse prevalence of its true class. There is no certain consensus in the literature about the last formula (the extension to the multi-class problem), so in this work it is used the same definition as in the `sklearn.metrics` Python's module.

Compared to accuracy, F_β-score gives a better insight into the functionality of a classifier, thanks to its β coefficient, being applicable to imbalanced data sets. The β value represents the ratio between the cost associated with predicting false negatives and false positives, which is derived from the fact that usually miss-classifying minority classes is more expensive than miss-classifying majority classes and so the F_β-score will be more (or less) affected when improving the recall rather than the precision for $\beta > 1$ (or $\beta < 1$) [54].

Moreover, it is also possible to adapt the macro-averaged F_β-score, F_{β_M}, in order to account for the data imbalance. This is called weighted F_β-score and it calculates the average of the classes F_β-scores, weighted by support, that is, the number of true predictions for each class:

$$weighted\ F_\beta = \frac{1}{\sum_{i=1}^{L} |y_i|} \sum_{i=1}^{L} y_i F_{\beta_i} \tag{3.22}$$

3.5.4 Clustering Performance Evaluation

Assessing how good is the performance of a clustering algorithm is not an easy task, at least is not that simple as evaluating a classifier's performance. In this case, instead of considering the cluster labels, it is evaluated the ability of separating the data into some meaningful subsets, that is, aggregating more similar observations in the same cluster and splitting the less similar ones into different clusters, based on some similarity measure. This section will only briefly introduce the basic concepts of clustering performance evaluation, since neither of the following stated metrics will actually be used for this work's particular problem. A better explanation of the reasons that led to this decision is in Chapter 4 (4.5.2).

The clustering evaluation metrics can be defined as intrinsic or extrinsic metrics. The intrinsic metrics measure how close to each other the observations from the same cluster are and also how distant they are from other observations in the other clusters. Conversely, extrinsic metrics compare the outputs of the clustering algorithm with some gold standard that is typically defined with human assistance and are usually used in the case of text clustering.

An example of intrinsic metric is the silhouette coefficient, based on both the mean distance between an observation and all other observations in the same category and also the mean distance between an observation and all other observations that are in the next nearest cluster. It is able to capture how dense and well separated clusters are, however, it is usually higher for convex clusters when comparing to, for example, density based clusters that are typically the ones obtained with DBSCAN algorithm [50].

Authors in [51] present a detailed study on extrinsic clustering evaluation metrics, starting by proposing a set of formal constraints that these metrics should satisfy and then analyzing some existent metrics based on these constraints. They define essentially four basic constraints as follows:

Homogeneity: the clusters must be homogeneous, that is, each cluster should not mix observations from different categories.

Completeness: the observations from the same category should be kept together in the same cluster.

Rag bag: it is better to have a disordered cluster among the others, with all the observations that cause disorder, than having disorder spread by all the clusters. This cluster is the “rag bag”, sometimes called “miscellaneous” or “other”, and contains all the observations of diverse categories that could not be assembled with other observations.

Size vs quantity: it is best to have a bigger cluster with small errors than to overly divide the data into several small clusters with minimal associated errors, which will probably lead to overfitting.

Since the extrinsic metrics presented by the aforementioned authors will not be directly used in this work, it is recommended to see [51] for details.

4 Implementation

Contents

The third Chapter explains the implemented computational experiments. It begins with some exploratory data analysis, all the way to each text preprocessing step, as well as the possible text representations and dimensionality reduction techniques used to prepare the data for the model. At last, the 3 possible scenarios for the model fitting (SSL, Clustering, and the combination of both of these strategies) are explained and a description of the proposed models setting for each scenario is made.

4.1 Dataset Description and Exploratory Analysis

Before proceeding with the description in the next sections of this chapter, the reader should know that all the exploratory data analysis, the pre-processing of the text data, and the implementation of the models were performed using the open-source programming language Python, through Google Colaboratory (Colab) notebooks, and some of its libraries, such as Pandas (`pandas`), Scikit-Learn (`sklearn`), SciPy, and others more text related such as Hunspell, NLTK and SpaCy [81–86].

4.1.1 Dataset

The original dataset did not contain all the components involved in the problem, actually, all the necessary data was spread across three different datasets. The main one was uploaded into a dataframe, `Data`, from the `pandas` library, that had 1110 entries and 17 columns, corresponding to some of the main components (response variables) considered in this problem, `L0`, `L1`, `L2`, and `L3`, their correspondent keys (`L0_KEY`, `L1_KEY`, `L2_KEY`, and `L3_KEY`), the `CAUSE_KEY`, other complementary components used to describe the scenario (besides `L0`), the `LINK`, and the correspondent `LINK_ID`.

Another dataset was provided, this time containing 2 columns, one with the `LINK` and the other with the `TEXT` associated with each link. It was also uploaded into a dataframe, `LinksData`, with 86239 entries, and all the empty (or with just one whitespace) texts were set to the `NaN` value. The idea was then to join this dataset with the previous one on the common variable `LINK`. Moreover, there was still a third dataset, also uploaded to a dataframe, `CausesData`, with 217 entries and 2 columns: `CAUSE_KEY` and `CAUSE`. It had the mapping between each `CAUSE_KEY` and the correspondent cause description, that is, the `CAUSE` itself.

In order to join all the information from the aforementioned datasets in only one final dataset, it was used the function `merge`, from the `pandas.DataFrame` library (similar to the SQL join). First, the dataset `LinksData` was merged to the `Data` on the common column `LINK` and in a ‘left’ manner, that is, using only the keys from `Data`. Then the exact same process was made but this time with the new and updated dataframe (already with the column `TEXT`), `Data`, and `CausesData`. The dataframe `Data` included now also the columns `TEXT` and `CAUSE`.

In addition, the original `LinksData` dataframe (not just with the unique values) was concatenated with the updated `Data`, with the function `concat` from the `pandas` library, that is, the pairs `LINK` and `TEXT` were appended to `Data` as new rows, with `NaN` values in all other columns. Also, some columns considered irrelevant for the analysis were deleted from the dataframe, namely, all the keys, identifiers (ID's), and the complementary components used to describe the scenario that had only one possible value besides `NaN`, and the rows that had `NaN` in the `TEXT` column were also removed.

Therefore, the final dataset that was going to be used in the following steps had 85032 observations and 9 columns (response variables): `L0`, `L1`, `L2`, `L3`, `ACCESSTYPE`, `SERVICETYPE`, `CAUSE`, `LINK` and `TEXT`.

Note that `ACCESSTYPE` and `SERVICETYPE` belong to the set of the client's scenario components, jointly with `L0` and thus they should represent only one sub-problem. However, henceforth only `L0` will be considered to the scenario's sub-problem. Also, the variable `LINK` will not be addressed, as it only serves the purpose of locating the `TEXT` in the troubleshoot guides website.

4.1.2 Exploratory Data Analysis

After performing the initial cleansing explained in the previous subsection, a first analysis on the response variables was made in order to study how many non `NaN` (missing) values existed in each `Data` column (count) and from those values how many unique values, which was the most frequent value (top) and how many times that value appeared (freq). The results were summarized in the following Table 4.1:

Table 4.1: Response variables description.

	count	**unique**	**top**	**freq**
L0	641	3	Voz	510
L1	607	14	One Net Telefone Fixo 3G	138
L2	567	39	Não recebe chamadas	121
L3	513	36	Todas as chamadas	183
CAUSE	630	92	Sem causa determinada	133

Then, each variable was studied individually, since each of them was being used as response variable for different sub-problems. In the graphs mentioned below it is possible to see the frequency distribution of each class, for each sub-problem. However, since the sub-problems `L2`, `L3` and `CAUSE` had several classes (15, 13 and 33, respectively) with only one sample (text) associated to them, these unique observations were not included in the plots, so that they could be better interpreted.

Observing the plots in Figures A.1, A.2, A.3, A.4, and A.5 (sited in Appendix A) it was noticed that all the sub-problems have imbalanced classes. Moreover, in sub-problems `L2`, `L3`, and `CAUSE` most of the classes had less than 10 observations (texts) associated to them, which is much less than the frequency of their majority classes presented in Table 4.1 (column freq).

The next step was to study the length of the texts, that is the number of words, in the dataset column

TEXT and some interesting insights came from this analysis. However, note that no text preprocessing was made until now.

First, it was generated a plot (Figure A.6 sited in Appendix A) with the possible text lengths and the number of texts that had each of those lengths (number of words). It could be observed that most of the texts had less than 200 words, the maximum text length was 557 words and the most frequent text lengths were 43, 100 and 24 words.

Then, after noticing that there existed texts with very few words, in fact, some with only one word, an important query has arisen: how many words should a text have in other to be relevant and worth to keep for the analysis? It was then observed that only texts with 3 or more words had at least a class associated, from one of the sub-problems labels. However, this was not enough as the unlabeled data was also going to be used for the categorization. Thus, it was generated a word cloud with the words that appear in those texts composed by only 1 or 2 words (Figures 4.1 and 4.2), to see if they could have some meaning inside the problem's scope.

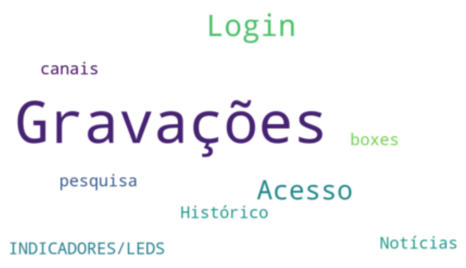

Figure 4.1: Wordcloud from texts with 1 word.

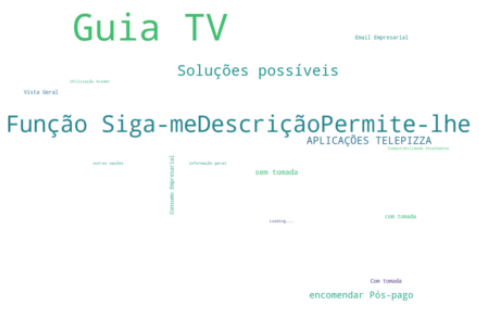

Figure 4.2: Wordcloud from texts with 2 words.

Based on Figure 4.2 it was decided to keep the 2 words from each text together in order to keep the meaning of the text. Both Figures 4.1 and 4.2 represent all the possible texts for each case, existing 9 unique texts with only 1 word and 16 possible different texts with only 2 words. As these words seemed to be meaningful, it was decided to keep these texts in the dataset.

At last, it was obtained a list of the 26 most common words across all texts in the dataset TEXT column, presented in Table 4.2. Again, this was the analysis before any text preprocessing and so it made perfect sense that the most common words were some of the Portuguese stopwords and also punctuation marks. This preliminary analysis on the text data led to some of the decisions taken for the text preprocessing steps explained in Section 4.2, e.g. removing punctuation and stopwords. It is curious to see that some of the words directly related with the problem's scope, even if before cleaning the text, already appear in this list such as "Cliente", "TV" and "Net".

Table 4.2: The 26 most common words and their frequencies (before text cleansing).

Word	Frequency	Word	Frequency
de	193409	no	39544
o	131791	se	39428
.	118840	da	36954
do	102205	não	36580
a	87141	para	34653
(	69826	é	30175
:	69114	Não	26789
)	66832	está	26604
e	57543	na	25618
que	55709	TV	25494
Cliente	51588	e-Phone	23681
com	43041	Net	23383
em	42079	por	23372

4.2 Text Preprocessing

4.2.1 Text Cleansing and Dictionary Construction

The first steps in the preprocessing phase consisted in cleaning the dataset. In the case of text based data this means removing irrelevant information, such as punctuation and other types of words/characters that were considered irrelevant to the problem in stake, namely, URLs, e-mail addresses, numbers, unusual characters or symbols not included in the punctuation set. Also, the resulting extra whitespace characters were removed.

After these preliminary steps, it was possible to start the dictionary construction. This is an essential part of this stage, since it will allow for text correction and help to narrow some possible errors when applying the classification model to it, for instance, due to misspelled words or several words with the same meaning or underlying idea.

It should be noticed that the texts in the dataset have a mixture of Portuguese and English words, being Portuguese the main one. For that reason, the Portuguese dictionary was chosen as a basis for the under construction dictionary, `my_dict`. Moreover, in order to avoid the spelling correction of the English words, it was defined a small auxiliary function with the aim of obtaining the set of English words and then add them to `my_dict`. Besides that, due to the nature of the problem, all the acronyms in the texts were also found and added to `my_dict`, as most of them were relevant. Just after that all the words in the text could be changed to lower case.

With the text cleaned and the dictionary built, it was defined a spellchecker (spelling corrector) that corrected each word of a text based on `my_dict`. Here it was used the Python package `hunspell`, that provides several functions for loading dictionaries, checking the spelling, adding words to a dictionary and getting suggestions for misspelled words. Thus, each word considered misspelled, that is, which did not belong to the list of words in `my_dict`, was replaced by the first word in the list of suggested words obtained with the `suggest` function from `hunspell`.

After a first spellchecker run against the texts in the dataset, several inconsistencies were found and thus a new analysis was done in an attempt to amend (a great part of) them. One of these issues was related with the fact that the hyphens were being removed along with other punctuation marks, even if they belonged to a word and were essential to its meaning (very common in Portuguese words, e.g. palavra-chave, wi-fi) and so it was applied a rule in order to maintain all the hyphens.

Another issue was the inconsistency when using the Portuguese Language Orthographic Agreement of 1990 [87], being the majority of the words written according to this agreement but some written according to the previous one. Since `my_dict` was based on the Portuguese dictionary after this agreement, some words were manually corrected to follow these rules and to avoid having a large number of different words that actually meant the same (e.g. "activar" to "ativar"). This led to other lines of thinking regarding the complexity of the Portuguese vocabulary, for instance, should all the words be rewritten using only one gender, as the female and male version of a word have the same meaning? However, this did not move forward as it was turning into a decision that could most likely cause some overfitting. Furthermore, some abbreviations and letters suppression were also amended (e.g. "nº" or "nr" to "número", "pt" to "portugal" and "1º" to "primeiro") as well as some typos found in the text.

Additionally, it was necessary to add other specific sets of words to the dictionary, such as a list of Portuguese first names, a list of Portuguese towns and villages and a list called "special words" that includes some telecommunications brands, problem specific words and other Portuguese and English words that, for some unknown reason, were being unnecessarily corrected.

After all these steps, the dictionary was finally built and all the texts from the dataset were corrected according to it, using the aforementioned spelling corrector.

4.2.2 Text Preprocessing

With all the text data cleaned it was possible to progress with the next preprocessing steps, that involved a more syntax related analysis but also a bit of semantic one.

The first step was to tokenize the text and then remove the Portuguese stopwords (e.g. a, o, e, um, uma, que, etc.), as they were not relevant for the text categorization. Here it was found an unsettling problem: the word "não", that gives negative connotation to its immediately following words, was being removed along with the other stopwords. This was an incorrect operation since, if the word "não" was removed, then a word and its negation would have the same meaning, which is not right. Hence, that word was removed from the Portuguese stopwords list and then a small function was implemented in order to join the word "não" with its immediately following word, connected by an underscore, wherever it was found. After this, the tokens were joined again into a continuous text, as in the following step the text should not be already tokenized.

Next, the lemmatization step was started. For that purpose, it was necessary to found a Portuguese dataset for NLP that was already trained by a model, able to identify each word in the text as a noun, an adjective, a verb and so on, so that the lemmatization could be then applied. Since there are still not many options for the Portuguese language, the best dataset found generated quite bad results, such as mixing up nouns and adjectives. Thus, it was decided to only apply the lemmatization to the text verbs, being defined a function in order to identify these words and then apply lemmatization to them. After applying this function to the whole text dataset, the preprocessing stage was concluded.

4.2.3 Solutions for The Imbalanced Dataset

In the dataset description it was stated that this dataset was clearly imbalanced for all the sub problems to be explored. In an attempt to solve this issue, some known techniques were applied.

The simplest ones were the over sampling, that replicated some of the observations from the minority class, and the under sampling, that removed some of the majority class observations. Note that this techniques should only be applied to the training set, as they intend to influence the fit of the classification models and, in this specific problem, only to the labeled subset (since the training set includes both labeled and unlabeled observations). There is a parameter called `sampling_strategy`, common for both techniques, that determines which targeted classes should be resampled and it can be defined by the desired ratio of the number of samples in the minority class (after resampling, in the case of over sampling) over the number of samples in the majority class (after resampling, in the case of under sampling), by setting the desired number of samples for each targeted class or by specifying the targeted classes. For the latter, the over sampling options are all, only the minority, all but the majority (the default option), all but the minority and the under sampling options are all, only the majority, all but the minority (the default option), all but the majority.

The SMOTE technique, similar to the previous techniques, could not be applied to the original dataset, since it does not work in the presence of classes with only one observation. After noticing this drawback, the thought of replicating the classes with only one observation started to make less sense as the information on those classes was scarce.

Therefore, two new lines of thought were explored. First, the idea was to remove the unique observations (classes with only one observation), but in this way part of the information would be lost and the results were not better. Then, it was noticed that one of the problems was that most of these unique observations were being used in the test set and not in the train set. Thus, the following strategy was to split the data into train and test sets, without the unique observations (being possible to use the `stratify` attribute of the function `train_test_split` from the `sklearn.model_selection` library) and then add these observations to the train set, so it could have more examples. However, note that this could lead to overfitting, which was a risk consciously taken.

4.3 Text Representation

As already explained in Section 2.3, in order to apply ML algorithms to this particular dataset, the words in each text will need to represented by numerical values, and so the tasks outlined below were performed with that exact purpose.

The first two techniques implemented were the BoW, in this case with term frequency (TF, hereafter), and the TF-IDF. In both of them a matrix, in which the rows represent the dataset texts (observations) and the columns its terms (features), was generated. Specifically, in the problem at hand, the terms were chosen to be the text single words. The entries of the matrix correspond to the number of times each word occurs in the document of that particular row, in case of the TF, or the TF-IDF weight of the word for that document, in case of the TF-IDF model, that is, taking into account the number of times that word occurs in the whole set of texts. The Python function used was the `TfidfVectorizer`, from the sklearn `feature_extraction` library. Note that this function has a default input parameter that will smoothen the idf component of each word, w, as follows (see (2.1) to compare with the original formula):

$$idf(w) = \log\left(\frac{1+m}{1+df(w)}\right) + 1 \tag{4.1}$$

Moreover, each matrix row (or TF-IDF vector) is normalized to have unitary Euclidean norm (L2 norm), that is:

$$v_{\text{norm}} = \frac{v}{\|v\|_2} = \frac{v}{\sqrt{v_1^2 + v_2^2 + \cdots + v_n^2}} \tag{4.2}$$

With the aim of being able to include the similarity between the words in each text and in that way trying to get some meaning out of them, there were applied other kinds of text representations, such as Word2Vec (W2V), Doc2Vec (D2V) and BERT.

For W2V it was necessary to define a vocabulary, that is, the set of words that would be used to define the context of each word in its vector representation. This could be done either by using an existing documents dataset that is already pre-trained in a word embedding model (e.g. Google News dataset, with about 100 billion words) or by using all the existing words in the set of documents and then train the word embedding model based on the similarity between each pair of the vocabulary words. Since there are still no sufficiently good pre-trained models for the Portuguese language and the pre-trained models may not contain specific words from the application domain being used, it was decided to use the latter option, even though it is associated with a higher computational complexity.

After training the word embedding model with W2V (from `Gensim` Python library), it was obtained a matrix in which the rows corresponded to the feature vectors of each word and the number of columns was the chosen size, that is, the dimensionality of the feature vectors. With this vocabulary table it was possible to find, for example, the similarity between each pair of words or the most similar words to a specific word, however, it was necessary to pass these scores to the feature matrix used as input for the text categorization, as the one produced by the TF-IDF. Thus, a range of possibilities was tested.

The most obvious decision was to use D2V (also from `Gensim` Python library) instead of W2V algorithm. D2V is just an extension of W2V, that instead of computing embedding vectors for each word does that for each document. The simplest form of this algorithm takes the W2V vectors of every word in the text and then aggregates them together by taking a normalized sum or an arithmetic mean of the terms. While adding the word vectors together it works as a sort of random walk, where some of the words will cancel each other (noise) and the direction of the final vector will be drifted towards that document topic direction.

Another approach was to take the mean of each word vector representation obtained with W2V, use it as a weight for that word and then replacing the TF-IDF weights for the W2V ones in the correspondent words defined in the feature matrix built with the former method.

In both cases, several parameters needed to be evaluated in order to find the combination that would lead to the best results. The first thing was to choose the dimensionality of the word vectors, then the algorithm to be used, CBOW (distributed bag-of-words in D2V) or skip-gram (distributed memory in D2V) and then whether or not to scale the values between 0 and 1.

Afterwards, the BERT method was also implemented and tested. Here it was not an option to train a model from scratch due to the high complexity and computational cost of that operation. Therefore, a Portuguese pre-trained BERT model had to be found. Unfortunately, there is a lack of Portuguese models, however, [18] proposed one pre-trained model for Brazilian Portuguese, BERTimbau, and so it was the one used in this work. After downloading this pre-trained model, the process consisted in a sequence of steps explained below.

First, a tokenizer was defined, in order to be able to include the special tokens, `[CLS]` and `[SEP]`, that define the beginning and end of a sentence or sequence of tokens, or also the parameter for the maximum length of each sequence that is used if the truncation is set to `True`, truncating each sequence of tokens to the defined maximum length. Still using this tokenizer, the tokens (words) are replaced by their token id's (numbers).

Then padding should be applied, or in other words, all the sequences of tokens should be set to the same size. In this way, extra indexes will be added to the shorter sentences, that will have the token id 0 in the "blank" spaces until the maximum length defined, whether by the length of the longest sequence or by the value defined for the truncation when setting the tokenizer, is reached. This step is performed with the aim of speed up the calculations since, after tokenizing the set of sentences, these are represented by a list of lists of tokens with different sizes and, with the padding, it is possible to represent these sequences of tokens as a 2D array. Moreover, an attention mask had to be created, so that the BERT model could ignore the padding whilst processing the input, otherwise it could slightly confuse it.

With the input already set, a tensor was built, that is, a multidimensional matrix defined by the number of sentences, the number of tokens (maximum length) and the number of hidden layers of the model (768 for the base model and 1024 for the large version), for both the padded and the masked inputs.

Finally, pre-trained BERT model was applied, using these last defined tensors as input. The output was a vector with dimension 768 (or 1024) for each token but only the first vector, corresponding to the initial special token `[CLS]`, was considered since BERT uses that token in a sentence classification way and so it can be used as an embedding for the entire sentence. Hence, the feature matrix obtained had one row per each document (sentence) with dimension 768 (or 1024) and it was ready to be used in the text categorization.

Although BERT seemed to be the best and most complete option for text representation, it turn out to be even more computationally intensive than it was expected, and so it was not possible to run it for all the documents in the dataset (only 6% of the observations were used and even in this case it was very time consuming). Thus, it was decided not to include this method in the experiments.

4.4 Dimensionality Reduction

Despite some of the words had already been removed (e.g. stopwords) in the preprocessing and even the text representation stages, there was still a quite high number of features. Being so, new attempts to reduce the text categorization and consequently, model complexity, were tried in order to improve the results, through dimensionality reduction, either by feature extraction or feature selection.

The methods used for feature extraction, that built a new and smaller set of features based on the original ones, were PCA, LSA and LDA. The process was very similar for each method: define the feature extraction method, fit the classification model based on the train set and then apply the fitted model to both train and test sets. Although these algorithms have some parameters to be set, the `n_components` was the one that was tested for different values, until the best result was reached.

When using the function `PCA`, from `sklearn`, it was sometimes necessary to first scale the data, depending on the text representation used (e.g. with TF-IDF it was not necessary as the data was already scaled). Moreover, in this method it was possible to choose the number of components to keep, `n_components`, based on the amount of variance that needed to be explained, e.g. 95%, and not just by the number of components itself as in LSA or LDA. In fact, after applying `PCA`, it is possible to know how many components were used in order to have the desired $\%$ of variance explained, and that number could be used as a starting point for the parameter `n_components` in the other two methods.

The `TruncatedSVD` algorithm from the `sklearn` Python's library was used for the LSA method, since they work in the same way on term count or TF-IDF matrices. In addition, `LatentDirichletAllocation`, also from Python's `sklearn` library, was used for the LDA method. On one hand, PCA and LSA had very similar (or even the same) results when using the same number of components and were both very fast. On the other hand, LDA took a lot more time to train its model and led to much worse results when using the same number of components (less 20% of accuracy). However, after reducing considerably, yet not too much, the number of components, the results were almost as good as with the previous algorithms (these values are specified in Chapter 5).

For the feature selection methods it was used the `SelectKBest` function from the `sklearn` library. It has two input parameters, `score_func`, that defines which function will be used to compute the scores for each feature, and `k`, that corresponds to the number of top features to select. In this case, 'chi2' and 'mutual_info_classif' were used to compute the scores for the Chi-Square (Chi2) and MI methods, respectively.

4.5 Model Fitting

First of all, even though it could only be applied to the labeled data, it was decided to use the NB classifier as the baseline model for this problem, since it is the simplest method and, despite not knowing if its conditional independence assumption is satisfied by the dataset, it is known that this algorithm still produces quite good results even if the condition is not satisfied. Since `MultinomialNB` implements the NB algorithm for multinomial distributed data (and in practice works well with TF-IDF feature matrices), it seemed to be the most appropriate function, from the `sklearn.naive_bayes` library, to use. The only manipulated parameter was `alpha`, the additive (Laplace/Lidstone) smoothing parameter.

From there on, the goal was to find a model that could obtain better results and, if possible, use the whole dataset (including the labeled and unlabeled observations). Given the nature of the problem (many unlabeled observations and few labeled ones) and with the goal of classifying the dataset texts into different classes, for each sub-problem, the first though was to implement SSL techniques.

The first obstacle found was regarding the high computational capacity required in order to run all the 85032 observations of the dataset. Unfortunately, even using Colab, in which the notebooks run on Google's cloud servers, where GPUs and TPUs can be used in order to boost the power of Google compute engine and where there are 25 GB of RAM available, it was not possible to run the whole dataset when using the SSL methods. The first attempt was to perform the classification by batches, however, it was not possible to do so since the required `partial_fit` functionality could not be applied to the SSL algorithms that were used. In this way, when splitting the data into training and test sets, it was first performed the split only with the labeled subset and then a percentage (no more than 35%) of the unlabeled subset was added to the train set.

Moreover, probably related with the considerable imbalance between the classes in the labeled set and also the quite large amount of different possible labels, the first results were not as promising as expected. In fact, the unlabeled data seemed to “harm” the results or not having influence at all.

Later on, the final goal initially thought for this project had to suffer some changes. Since it was not possible to obtain sufficiently good accuracy results when trying to classify the texts into a specific label, the aim was then to classify them into a small set of labels. Hence, in the next sections three possible scenarios are explained. The scenario 1 uses only the SSL techniques, the scenario 2 mixes both unsupervised and SSL techniques and the scenario 3 uses only the unsupervised learning techniques. The last scenario aimed to understand if the SSL stage used in scenario 2 was not a redundant step. Note that with this option it is not possible to use the classification results to construct the initially required paths in the inverse direction, that is, the sequence of events from the first sub-problem until the last one and the text that results from that (helping the technical assistant to solve the client’s complaint). Nevertheless, this is a starting point that could be used to narrow the currently existing wide range of possibilities (choosing between 10 possibilities is always better than choosing between 93) and, after a proper analysis from a specialized technician, be used to the initial objective.

Another matter that needed to be better explored was the fact that the sub-problems were dependent on each other and should be linked in a specific order. Therefore, in order to include the information of the precedent sub-problems it was decided to include the name of the labels provided by each of those sub-problems as words (features) in the texts of the current sub-problem. For instance, when classifying the texts into the sub-problem `L1`, the known `L0` labels associated with those observations should be appended, as a new word, to the texts used as the input dataset and this should be repeated for each dataset observation (text). For the classification regarding the `L2` sub-problem, both `L0` and `L1` should be added to the input texts, and so on, until the last sub-problem, related with the `Cause` labels, is reached.

4.5.1 Scenario 1: Semi-Supervised Learning

Although there exist several SSL algorithms described in the literature, one of the biggest issues regarding this scenario was the fact that most of them were not yet implemented in the known Python libraries. However, there are some experiments shared in the GitHub open source community and thus some of them were used throughout this work.

The main explored algorithms were `LabelPropagation` and `LabelSpreading`, from the `sklearn` Python’s library, and then `SelfLearningModel` from `https://github.com/tmadl/semisup-learn` and `Semi_EM_MultinomialNB` from `https://github.com/jerry-shijieli/Text_Classification_Using_EM_And_Semisupervied_Learning`. The former link also had another SSL algorithm implemented, Contrastive Pessimistic Likelihood Estimation (CPLE), however, after some unsuccessful attempts, it was assumed that it could not be applied to the problem in question.

The parameters adjusted in `LabelPropagation` and `LabelSpreading` were the following:

- `kernel:` Kernel method used to construct the similarity graph. The possible options are 'knn' or 'rbf'. Note that, while the rbf kernel produces a fully connected graph, the knn kernel produces a sparse matrix. Thus, with very large input matrices, rbf may lead to prohibitively long running times and high cost of calculation and in the case of knn, conversely, since the produced sparse matrix is much more memory-friendly, the running times are usually significantly reduced.
- `gamma:` This parameter is used for rbf kernel.
- `n_neighbors:` This parameter specifies the number of neighbors to be considered by knn and needs to be strictly positive.
- `max_iter:` Maximum number of iterations allowed (default $= 1000$). If it reaches this number the model stops, even if convergence was not yet reached.
- `n_jobs:` Number of parallel jobs to run or the maximum number of running workers the engine is allowed to use. `None` means only one is used and -1 means all processors (CPUs) are being used.
- `alpha:` This is the clamping factor and is only used for `LabelSpreading` (it is always zero for `LabelPropagation`). It is a value between 0 and 1 that defines the percentage of information that should be adopted from the neighbors to the label distribution (the rest is retained from the original label distribution), that is, how much the confidence of the distribution can be changed.

In the case of `SelfLearningModel`, there are only 3 parameters:

- `basemodel:` The classification model that will be iteratively self trained. It can be, for instance, `MultinomialNB`, `SVC` or `SGDClassifier`.
- `max_iter:` Maximum number of iterations allowed (default $= 200$).
- `prob_threshold:` Probability threshold (default $= 0.8$) for self-labeled observations (the confidently self-labeled observations are the ones with above-threshold probability).

Only the parameter `basemodel` can be changed when defining the model, the others are intrinsically defined and can only be changed in the code that defines the class `SelfLearningModel`.

The `Semi_EM_MultinomialNB` algorithm uses the default values of the `MultinomialNB` input parameters and it also uses the `max_iter`$= 30$ and `tol`$= 10^{-6}$ (convergence tolerance) parameters to define when a steady state is reached. If it is necessary to change these parameters, it should be done in the code that defines the class `Semi_EM_MultinomialNB`.

In order to select the best set of parameters for LP and LS, the initial idea was to perform k-fold cross validation with the sklearn function GridSearchCV. The problem was that this could only be done using the labeled data due to the splitting into the k folds, otherwise the test set would have unlabeled samples in some of the splits. Hence, this was performed as if it was a supervised learning model. Noticing that the results did not seem completely accurate, probably related with the fact that several classes had only one observation assigned to them, there was the need to test each possible set of parameters individually and try to understand the effect that each one of them had in the results. Essentially, it was tested the kernel function and the effect that its correspondent parameter (gamma for rbf and n_neighbors for knn) caused. This was also tested for different values of n_jobs and also for max_iter since a warning, saying that this number was reached without convergence, had arisen.

The workflow was defined by a set of consecutive steps. First, the vector of labels, y, had to be obtained from the principal dataframe, as this is different for each sub-problem, and then a label encoder was applied in order to transform the text labels into numerical ones. Plus, the unlabeled observations were set to the label '-1', since the SSL algorithms use this label as an identifier for the unlabeled observations. The feature matrix X was obtained from the column TEXT of the principal dataframe. It was initially always the same, although it was tested with different text numerical representations (being TF-IDF the main one), but then it became different for each sub-problem, as the previous sub-problems information (labels) was being included in the texts.

Then, the feature matrix and the labels vector were separated into labeled and unlabeled and a small percentage of the unlabeled data was set to be used further on. The labeled data was split into train and test sets (70% and 30% of the labeled dataset, respectively) and then the subset of unlabeled data obtained before was introduced in the train set. Finally, the SSL model was defined and fitted to the train set. With the test set, the model was tested and evaluated through a classification report, the accuracy and the weighted F_1-score (F_1-score hereafter).

4.5.2 Scenario 2: Clustering then Semi-Supervised Learning

The idea in this scenario was to first cluster the data, with the aim of defining new classes that are composed by a small set of the original classes, and then perform the SSL techniques already used in the first scenario, intending to obtain better results.

The clustering methods applied were KMeans and DBSCAN, from the sklearn.cluster library, since they are known to work well with a large number of observations, which is clearly the case in this problem, being K-means computationally faster than other algorithms and an easy to implement method. While in K-means it is necessary to define the number of clusters (parameter n_clusters) beforehand, in DBSCAN there is no need to define the number of clusters, instead, it is used the neighborhood size, defined by the parameters eps, the maximum distance between two samples for one to be considered as in the neighborhood of the other, and `min_samples`, the number of samples in a neighborhood for an observation to be considered as a core point (it includes the point itself). Moreover, higher values of `min_samples` or lower values of `eps` lead to higher density necessary to form a cluster.

Other parameters that had to be set to a different value than the default were the `algorithm` to 'full' (in the `KMeans`) since the default was only more efficient with well-defined clusters and this was most likely not the case due to the nature of the data, and the `metric` to 'cosine' (in the `DBSCAN`), since this is the most appropriate metric for calculating distance between text observations and the default was 'euclidean'.

Before moving on to the SSL stage, it was necessary to evaluate the performance of the clustering algorithms. The first attempt was to use known metrics such as the silhouette coefficient, the V-measure (requires the knowledge of the ground truth class assignments of the observations) or the Calinski-Harabasz index, however, this idea was almost instantaneously dropped as the results did not seem to be accurate. With all the above-mentioned metrics, the score seemed to constantly increase as the number of clusters was increasing. Therefore, it was necessary to built a specific performance metric that could capture the particularities of the given problem. Based on the basic constraints defined by [51] and analyzing the specificities of the problem, a set of parameters that should be taken into account in the under construction metric, was defined:

- **Number of clusters:** A smaller number of clusters is associated with a simpler model, yet this number being higher to the original number of classes is not an absurd idea in some particular contexts, which will be observed in the results.

- **Number of repeated clusters:** If there are too many repeated clusters, that is, clusters associated with exactly the same labels, it indicates that the data set is being excessively partitioned and this could lead to overfitting.

- **Number of clusters with more than 4 labels:** The number 4 was the initial choice, but then it was determined that this number could go up to 10. Here the idea was to narrow the hypothesis that would be later analyzed by the specialized technician, that was the main goal of this scenario.

- **Maximum cluster size:** This parameter is used in order to balance the previous one. If there are a lot of clusters with more than 4 associated labels but this number does not overcome a cap of 10 (or even 20, for the sub-problem `CAUSE`) then this is not problematic. On the other hand, if there are clusters with a considerable high number (e.g. more than 20) of associated labels it probably means that this is not the best cluster distribution.

- **Noisy observations (cluster '-1'):** Only relevant for the DBSCAN algorithm. It was first used to check if it existed the cluster '-1', which was initially considered as a drawback, since it meant that those observations and their associated classes were not being associated with any cluster. However, it was observed that the labels that were being attributed to this special cluster were exactly the ones with only 1 or 2 observations and so it was expected that they could not be correctly categorized in any of the other clusters.

In that way, with the parameters explained above, it was possible to test different combinations and find the best set of input parameters for each clustering algorithm, according to the problem properties. It should be noticed that in order to be able to evaluate the quality of the resulting clusters, it was necessary to use only the labeled training set, so that the ground truth labels were known and used in the aforementioned analysis.

With the best set of parameters chosen for the clustering method, it was explored what could be the best way to show the resulting clusters and organize them in order to use this information as input for the SSL phase. Based on an existing idea used in the KEEN project, return the set of labels of each new class (cluster) organized from the most likely label to occur to the least likely one seemed to be the most obvious thing to do. Aiming to compute this probability, the idea was to consider only the labeled training set and from that take into account not only the number of times that a label occurred in a particular cluster but also its total number of occurrences in the entire set of clusters, similarly to the TF-IDF logic. In fact, it could be used the TF-IDF scores formula but with the labels being the terms and the clusters being the documents, as a label frequency - inverse cluster frequency, multiplying the label frequency (number of times the label occurs in that particular cluster) by the inverse cluster frequency, defined by $log\left(\frac{1+\text{total nr of clusters}}{1+\text{nr of clusters that contain the label}}\right)+1$, and then normalizing by row with the L2 norm. This was not fully implemented and thus it will be suggested as future work.

Summing up, the workflow in scenario 2 consisted in the same process explained in scenario 1 but with the slightly difference of, after separating the data into labeled and unlabeled, defining the clustering algorithm and applying it to the labeled data. Then a new labels vector had to be built, where the new classes corresponded to the clusters (set of labels). After that, the process was the same, from the splitting into train and test data until the end.

4.5.3 Scenario 3: Clustering

After a quick analysis on the clustering methods available on `sklearn`, specially the ones that were being used in this experience, this scenario was only implemented for the `KMeans` algorithm, since it was the one in which the `predict` function could be directly applied. It could be useful, in future work, to compute the distances between the observations and the cluster centroids and use that information to classify (in the case of DBSCAN) each observation into the cluster that has the centroid with the smallest distance to it.

Moreover, only the labeled data was used in this scenario, since the labels were used to define the clusters and the set of labels that are associated to them. Plus, the split between train and test sets was made without the unique observations (that were afterwards added to the train set), so that the split between train and test sets could be balanced in terms of class representativeness. This idea of each set (train and test) containing approximately the same percentage of samples of each target class as in the complete set was then replicated in scenario 1 in order to improve its results.

Then the clustering model was defined and fitted to the train set, saving the information on which labels belong to each cluster. To evaluate the quality of the formed clusters it was used the same metric described in scenario 2. With this trained model the clusters of the test set were predicted and the attempt of evaluating the “classification” made consisted in verifying if the predicted clusters for the test set contained the actual label of each test sample. Knowing that, and attributing the value 1 when a match was found and 0 otherwise, it was possible to compute an accuracy score, summing all these values and dividing by the length of the test set. Although this is not theoretically correct, it was an approach used in order to understand if the SSL step from scenario 2 was actually relevant.

5 Results and Discussion

Contents

In this chapter the results are presented for each of the 5 sub-problems, analyzing the various applied approaches and comparing the different scenarios taken into account and the models used (introduced in the previous chapters). Then, these results are further discussed.

5.1 Results

Since the set of different approaches to test and compare, jointly with all the target values to analyze, lead to a wide range of possibilities, a set of base cases was first tested and then, with the best obtained results, different approaches for the dimensionality reduction and the text representation were considered and compared, for each model. The base cases consist of using TF-IDF for the text representation, not applying dimensionality reduction, unique observations included in the train/test split or included in the train set after the split (except for sub-problem `L0`). In all cases, it was used the whole labeled subset and 0%, 10% or 30% of the unlabeled subset, respectively (when applied).

Although all the possibilities were tested without including the labels of the previous sub-problems as words in the text, for sub-problems `L1`, `L2`, `L3` and `CAUSE`, the presented results only include the options where these words were included, since the dependence between the sub-problems is only reflected in this way (as explained in the Section 4.5). However, it should be referred that the results obtained with this condition were slightly worse (except for sub-problem `L1`). This is actually an expected outcome, since there are observations with the exact same text but with some of the sub-problems labels missing and therefore they are considered as different texts. Moreover, for the labels that are described by more than one word, the words are combined into only one word, joining them to the text without white spaces in between.

Since the k-fold cross validation could not be properly applied to the implemented models, the best parameters found in an exhaustive search are immediately presented, without comparing the possible options.

The evaluation metrics used are the method `score`, that returns the accuracy on the given test data and labels, and the `f1_score` from the `sklearn.metrics` package, with the parameter `average` set to *'weighted'*, which means that it calculates metrics for each label and finds their average weighted by support (i.e. the number of true observations for each label), accounting for label imbalance.

All the results will be compared with the baseline model, the NB classifier. This was the only case where the k-fold cross validation technique was applied in order to obtain the best value for the `alpha` parameter. It was used the `cross_val_score`, from the `sklearn.model_selection` library, with 10 folds and values between 0 and 1 for the additive smoothing parameter. The model obtained was the same used as the `basemodel` for the `SelfLearningModel`.

In order to perform the labeled data split into train and test sets, it was used the function `train_test_split`, from `sklearn.model_selection` package, with `random_state`$= 123$, so that the train and the test sets were always the same for each experience and the models could be correctly compared. Moreover, in the experiences where the unique observations were added to the train set after the splitting, the parameter `stratify` was used in order to have, for both train and test sets, approximately the same percentage of samples of each target class as the whole labeled subset (this cannot be applied when there exist unique observations).

Then, for the SSL algorithms, the unlabeled data was added to the train set. For `LabelPropagation` and `LabelSpreading`, it was used 0%, 10% and 30% of the unlabeled subset, whereas for `SelfLearningModel` and `Semi_EM_MultinomialNB` it was only used 10% and 30%, since for these last two algorithms it is mandatory to include unlabeled samples.

5.1.1 Sub-problem L0

The best set of parameters found for each model was:

- `MultinomialNB` and `SelfLearningModel`: `alpha`$= 0$, which means there is no smoothing.
- `LabelPropagation`: `gamma`$= 30$, `kernel`$=$ 'rbf', `max_iter`$= 100$, `n_neighbors`$= 7$
- `LabelSpreading`: `alpha`$= 0.15$, `gamma`$= 20$, `kernel`$=$ 'knn', `max_iter`$= 30$, `n_neighbors`$= 7$

This sub-problem has 641 labeled observations and the feature matrix dimensions obtained with the TF-IDF were 6075 words.

Table 5.1: Results for `L0` with TF-IDF.

Metric	*% unlabeled data*	**Naive Bayes**	**Self Learning**	**EM Algorithm**	**Label Propagation**	**Label Spreading**
Accuracy	0	0.9741			0.9793	0.9430
	10		0.8446	0.7979	0.9793	0.9275
	30		0.8446	0.8187	0.9793	0.9326
F1-score	0	0.9752			0.9792	0.9387
	10		0.8684	0.7082	0.9792	0.9266
	30		0.8684	0.7516	0.9792	0.9330

As it can be seen in Table 5.1, the results for the first sub-problem are quite good, as it was expected, since the texts are being classified into only 3 categories. Even though the LP model overcomes the NB classifier results, the improvement is less than 1%, which is not significant. Moreover, whilst for Self Learning and LP the unlabeled data seems to have no effect on the results, for the EM algorithm and the LS it can be seen that there is a slight improvement from the 10% to the 30% of unlabeled data used, although it is still worse than not using unlabeled data (for LS).

The Self Learning and the EM algorithm results are not bad, however, when compared to the others, are significantly worse, with less than 90% accuracy/F_1-score. On the other hand, all the other models have quite good results and hence no dimensionality reduction technique was applied. However, these models were also tested with other text representations, as it is described in Table B.1 (sited in Appendix B). The results were, in general, worse, being TF the one with closest results to TF-IDF (as expected) and W2V and D2V actually improving with the increase of the amount of unlabeled data used but still considerably worse than TF. In particular, this time LS had much better results than LP, that in turn had substantially worse results (more than 10% lower).

Since the feature matrix generated with the W2V weights had some negative entries, it had to be scaled to 0-1 before applying the NB classifier (as it does not work with variables that assume negative values), which could be the reason for worse results. Moreover, with the D2V representation it was necessary to reduce the dimensions of the vocabulary used (to 460 features) in order to be able to run the models in a reasonable amount of time and without exceeding the RAM limit.

One issue that could only be spotted when analyzing the `classification_report` of each model, was the fact that LP was only able to correctly classify the majority class, which is clearly masked by TF-IDF and stands out with the other text representations.

5.1.2 Sub-problem L1

The best set of parameters found for each model was:

- `MultinomialNB` and `SelfLearningModel`: `alpha`$= 0.11$
- `LabelPropagation`: `gamma`$= 20$, `kernel`$=$ 'knn', `max_iter`$= 100$, `n_neighbors`$= 7$
- `LabelSpreading`: `alpha`$= 0.15$, `gamma`$= 20$, `kernel`$=$ 'rbf', `max_iter`$= 30$, `n_neighbors`$= 7$

This sub-problem has 606 labeled observations and the feature matrix dimensions obtained with the TF-IDF were 6075 words, the same has in sub-problem `L0`, which was expected since the words (correspondent to the 3 labels from `L0`) added to the vocabulary were most likely already present in it.

Looking at Table 5.2 it is possible to infer that the LS is the only model that overcomes the baseline model, LP obtains very close results and Self Learning and EM algorithm obtain quite bad results, having the latter accuracy and F_1-score lower than 50%. Thus, the following analysis will only consider the LP and LS models.

Moreover, note that, in this case, adding unlabeled data actually worsen the results for LP and LS. In LS the results with unlabeled data are always the same (up to 4% worse), regardless of the amount, but in LP it is always decreasing with the increase of unlabeled data used (up to 15% worse). Self Learning is the only algorithm that actually improves with the increase in the amount of unlabeled data (up to 10%).

Table 5.2: Results for `L1` with TF-IDF and no dimensionality reduction.

Unique samples	*Metric*	*% unlabeled data*	**Naive Bayes**	**Self Learning**	**EM Algorithm**	**Label Propagation**	**Label Spreading**
Included in train/test split	**Accuracy**	0	0.7596			0.7322	0.8087
		10		0.5574	0.4754	0.6393	0.7650
		30		0.6339	0.4645	0.5847	0.7650
	F1-score	0	0.7143			0.7009	0.7877
		10		0.5854	0.3596	0.6462	0.7543
		30		0.6331	0.3472	0.5897	0.7543
Included after train/test split	**Accuracy**	0	0.7802			0.8077	0.8462
		10		0.6044	0.4451	0.6538	0.8242
		30		0.7033	0.4286	0.6374	0.8242
	F1-score	0	0.7431			0.7888	0.8315
		10		0.6154	0.3537	0.6800	0.8165
		30		0.6811	0.3256	0.6479	0.8165

On the other hand, as expected, including the unique observations after the train/test split improves the results (up to 8%), except for the EM algorithm. In fact, both LP and LS overcome the NB classifier results, by 2%-4% (accuracy/F_1-score) and 6%-9%, respectively.

Although the results were good when including the unique samples after the train/test split, they were still below the 90% accuracy/F_1-score. Therefore, attempting to improve these results, the five dimensionality reduction techniques described in Section 2.4 were applied and the Table B.2 (sited in Appendix B) contains the comparison between them. An important aspect to consider is that, when applying these techniques to the train and test sets, the feature selection (Chi2 and MI) is faster than the feature extraction (PCA, LSA, LDA), however, when training the model, the latter is faster.

First, PCA was applied, since it is the only technique that enables selecting the number of components to keep based on the amount of variability that should be explained, in this case defined as 95%. However, the number of components obtained with this condition was different for each amount of unlabeled data, that is, 57, 931, and 460 number of components for 0%, 10%, and 30% of unlabeled data, respectively. This is, in part, related to the fact that the number of components in PCA cannot be higher than the number of samples, which does not apply to the other techniques.

In this sub-problem, these values were used as well for the LSA, which will not be the case for the following ones, that will always use 931 components, as it is the value that leads to better results when compared to the model without dimensionality reduction. Note that PCA and LSA produce very similar results and, with 931 components (and 10% of unlabeled data), LSA results are equal or higher than the PCA ones, being the latter equal or slightly higher when using other set of parameters.

Another relevant aspect is that training LDA is very time consuming, specially when a high number of components is defined and, moreover, the results are actually worse. Thus, for this technique 95 components was the number that generated best results with reasonable computation times. However it still took a lot of time training this feature extraction model and, as it can be seen in Table B.2 (sited in Appendix B), the results were not better and, in the case of LS, the results were substantially worse (from 10% to almost 20% worse than the original model). On the other hand, it is interesting to notice that the LDA results improve when using 10% of the unlabeled data, whereas with the other techniques the results decrease with the increase of the unlabeled data used.

The feature selection techniques have similar results to the previous mentioned ones and both lead to the exact same values. Since Chi2 does not work with features that assume negative values, the MI will be the only one applied hereafter. In general, when compared with the original model, some of the combinations presented in Table B.2 (sited in Appendix B) have better results but this improvement is no more than 2%.

The next step was to try different text representations, comparing the results between the baseline model, LP and LS and this time only MI, LSA as the dimensionality reduction techniques, since they were the only ones without drawbacks.

Observing the results with the TF text representation, in Table B.3 (sited in Appendix B), it is possible to see that the dimensionality reduction techniques lead to similar or even slightly worse results than using the original model. It is important to refer that, attempting to obtain the best results, the NB classifier used $\alpha = 0$ (instead of 0.11, that with this text representation had worse results) and the number of components used for LSA was 460, although all the other models used 931. The number of components used for MI was always 931. In order to simplify the comparisons, these parameters will be equally set in the following sub-problems.

It is interesting to see that, in general, whereas the baseline model and LP have better results with this text representation (more 6% to 8% or 1% to 5% of accuracy/F_1-score, respectively), LS has much worse results, specially when increasing the amount of unlabeled data. This also happens with W2V text representation in Table B.4 (sited in Appendix B), however, with even worse results for LS and no significant improvements for the other two models. On the other hand, there was an improvement when using dimensionality reduction techniques.

Again, the feature matrix generated with the W2V weights had to be scaled to 0-1 before applying the NB classifier, which could be the reason for worse results. Moreover, using this matrix required a higher amount of RAM and so it was not possible to run the SSL models with 30% of unlabeled data. Therefore, this text representation will not be considered in the next sub-problems.

In the case of D2V representation (Table B.5, sited in Appendix B), no dimensionality reduction technique was applied, since the dimensions of the vocabulary used were again reduced (to 460 features). Maybe for that reason the results did not improve and, in this way, this text representation will not be used in the following sub-problems as well.

5.1.3 Sub-problem L2

The best set of parameters found for each model was:

- `MultinomialNB` and `SelfLearningModel`: `alpha`$= 0$, which means there is no smoothing.
- `LabelPropagation`: `gamma`$= 35$, `kernel`$=$ 'rbf', `max_iter`$= 100$, `n_neighbors`$= 7$
- `LabelSpreading`: `alpha`$= 0.15$, `gamma`$= 20$, `kernel`$=$ 'rbf', `max_iter`$= 30$, `n_neighbors`$= 7$

This sub-problem has 552 labeled observations and the feature matrix dimensions obtained with the TF-IDF were 6090 words.

For the reasons stated above, the following sub-problems will be only tested with the TF-IDF text representation and the dimensionality reduction techniques, if applied, will only be MI and LSA, always including the unique observations in the training set after the train/test split.

Table 5.3: Results for L2 with TF-IDF and unique observations included after train/test split.

Metric	*% unlabeled data*	**Naive Bayes**	**Self Learning**	**EM Algorithm**	**Label Propagation**	**Label Spreading**
Accuracy	0	0.5120			0.5542	0.5482
	10		0.4940	0.3373	0.5422	0.5422
	30		0.4940	0.3554	0.5422	0.5482
F1-score	0	0.5121			0.5501	0.5316
	10		0.4992	0.2325	0.5342	0.5187
	30		0.4992	0.2429	0.5342	0.5217

As it can be observed in Table 5.3, the results are quite bad in general. Self Learning and EM algorithm have accuracy/F_1-score values below 50% and the others do not have much higher values, however, both LP and LS better results than the baseline model. The EM algorithm slightly improves when using more unlabeled data, whereas Self Learning has always the same result. On the other hand, LP and LS have slightly worse results with the increase of the amount of unlabeled data used. Moreover, for LS it is better to use 30% than 10% of the unlabeled data, whereas for LP it is the same.

The LP and LS models were tested after applying dimensionality reduction techniques but, as there was no significant improvement, these results will not be addressed.

The results of this sub-problem are most likely related with the imbalance between classes but also with the fact that there are 39 possible classes. Hence, aiming to solve this issue and obtain better results, it was followed the approach from the scenario 2 (see Subsection 4.5.2), applying it only to LP and LS. The first thought was to use a number of clusters smaller than the number of classes. With the K-means clustering algorithm, despite the very outstanding accuracy/F_1-score results when evaluating the models, after using the metric explained in Subsection 4.5.2 to evaluate the clustering, it was clear that this would not be a clever idea, since there were clusters with 29 labels associated, which is not very helpful.

Then, it was done an analysis in order to find the best number of clusters according to that metric (with the labeled set) and to the accuracy/F_1-score results then produced with the SSL models (and their new labels). The best solution found was to use 65 clusters, since although 11 of them were repeated, only 9 had more than 4 labels associated and for those, the maximum number of labels associated was 6. The results were the same for both LP and LS, having accuracy$= 0.9766$ and F_1-score$= 0.9742$ without unlabeled data and accuracy$= 0.9649$ and F_1-score$= 0.9610$ (or 0.9602) with 10% or 30% of it.

For the DBSCAN algorithm it was done a similar analysis but varying two parameters (`eps` and `min_samples`). It was verified that a higher value of `min_samples` leads to a smaller number of clusters but increases significantly their maximum length. On the other hand, a smaller `eps` value improves the results but in return increases the number of clusters and their maximum length (which also happens with `eps` values higher than a certain threshold). Hence, the best set of parameters found for this sub-problem was `eps`$= 0.65$ and `min_samples`$= 2$ (which corresponds to 42 clusters with a maximum length of 10 labels). Note that with these parameters, the noisy cluster ('-1') exists and it was observed that the labels with only 1 sample were the ones attributed to it, which makes perfect sense.

Again, the results were equal for LP and LS, being accuracy$= 0.9762$ (regardless the amount of unlabeled data) and F_1-score$= 0.9722$ with no unlabeled data and F_1-score$= 0.9704$ with 10% or 30% of it. Thus, although DBSCAN gets a better F_1-score (although the accuracy is slightly worse, the difference is less than 0.05%) with a lower number of clusters, K-means has a lower maximum cluster length. These results were actually expected, since in this scenario the cluster assumption is "forced" to be satisfied, as the similar labels are being organized in the same clusters a priori.

Moreover, as the unlabeled data seemed not to have a significant influence in the results (actually, it slightly worsen them), the scenario 3 (see Subsection 4.5.3) was also tested, that is, using only the labeled data and applying only the clustering algorithm. Note that this could only be done with the K-means algorithm, in order to be able to evaluate the model as if it was a classification problem (as explained in Subsection 4.5.3). The score obtained was 0.9157, worse than it was obtained in the scenario 2, which could confirm that the latter was a better option. Note, however, that the evaluation metric is not exactly the same and so this decision is not straightforward.

5.1.4 Sub-problem L3

The best set of parameters found for each model was:

- `MultinomialNB` and `SelfLearningModel`: `alpha`$= 0.44$
- `LabelPropagation`: `gamma`$= 300$, `kernel`$=$ 'rbf', `max_iter`$= 30$, `n_neighbors`$= 3$
- `LabelSpreading`: `alpha`$= 0.15$, `gamma`$= 20$, `kernel`$=$ 'rbf', `max_iter`$= 30$, `n_neighbors`$= 7$

This sub-problem has 502 labeled observations and the feature matrix dimensions obtained with the TF-IDF were 6152 words.

Table 5.4: Results for L3 with TF-IDF and unique observations included after train/test split.

Metric	*% unlabeled data*	**Naive Bayes**	**Self Learning**	**EM Algorithm**	**Label Propagation**	**Label Spreading**
Accuracy	0	0.3709			0.3974	0.4238
	10		0.3709	0.3576	0.3974	0.4238
	30		0.3709	0.3576	0.3974	0.4172
F1-score	0	0.2386			0.3825	0.3805
	10		0.2386	0.1967	0.3832	0.3778
	30		0.2386	0.1967	0.3832	0.3757

From Table 5.4, it can be seen that the results are even worse than the ones from the previous sub-problem, this time with all models having accuracy/F_1-score below the 50%. Therefore, moving on to the scenario 2 approach was a straightforward decision.

In this case, the best solution found for the K-Means algorithm was to use 50 clusters (with maximum length of 8 labels) and the best set of parameters found for DBSCAN was $\texttt{eps}= 0.5$ and $\texttt{min\_samples}= 2$ (which corresponds to 49 clusters, with the noisy cluster included, and a maximum cluster length of 8 labels). The results were again quite good, being slightly better with the K-means. The LP had always the same accuracy and F_1-score, 0.9935, regardless the amount of unlabeled data, and the LS had always accuracy$= 0.9870$ and then F_1-score improving from 0.9810 to 0.9870 with the 10% or 30% of the unlabeled data. With the DBSCAN, LP got accuracy$= 0.9737$ and F_1-score$= 0.9683$ (or 0.9680, with unlabeled data) and LS obtained slightly better results, namely, accuracy$= 0.9803$ and F_1-score$= 0.9745$ (or 0.9725, with unlabeled data). Hence, in this sub-problem, for the scenario 2, it would be best to use the K-means algorithm.

Similarly to the sub-problem L2, again for the same reasons, the scenario 3 was also tested and the score obtained was 0.9139.

5.1.5 Sub-problem CAUSE

The best set of parameters found for each model was:

- `MultinomialNB` and `SelfLearningModel`: $\texttt{alpha}= 0.78$
- `LabelPropagation`: $\texttt{gamma}= 5$, $\texttt{kernel}=$ ‘rbf’, $\texttt{max\_iter}= 1000$, $\texttt{n\_neighbors}= 7$
- `LabelSpreading`: $\texttt{alpha}= 0.15$, $\texttt{gamma}= 20$, $\texttt{kernel}=$ ‘knn’, $\texttt{max\_iter}= 30$, $\texttt{n\_neighbors}= 7$

This sub-problem has 597 labeled observations and the feature matrix dimensions obtained with the TF-IDF were 6376 words.

Table 5.5: Results for `CAUSE` with TF-IDF and unique observations included after train/test split.

Metric	*% unlabeled data*	**Naive Bayes**	**Self Learning**	**EM Algorithm**	**Label Propagation**	**Label Spreading**
Accuracy	0	0.4889			0.4278	0.4111
	10		0.3778	0.2222	0.4167	0.2556
	30		0.3778	0.2333	0.4333	0.2333
F1-score	0	0.4205			0.3993	0.3528
	10		0.2702	0.0808	0.3768	0.2417
	30		0.2702	0.0958	0.3818	0.2170

Looking at Table 5.5 it is possible to observe that, even though this sub-problem has almost three times more labels than in `L3`, the results are actually quite better, with the exception of EM algorithm and LS. Nevertheless, the accuracy/F_1-score were still below the 50% for all the models and so the approach from scenario 2 was also applied, noticing that this time the results were not as good as in the last two sub-problems. Moreover, in addition to the usual improvement in the results of the EM algorithm with the increase in the amount of unlabeled data, the LP results also have a slight increase when using 10% to 30% of unlabeled data.

Starting with the K-means algorithm, the number of clusters used was 65 (with maximum cluster length of 11 labels). Note that, for the first time, the best number of clusters found was lower than the number of original labels. Both the accuracy and the F_1-score had worse results with the increase in the amount of unlabeled data used, for LP and LS. The former went from accuracy$= 0.9788$ and F_1-score$= 0.9779$ to accuracy$= 0.9259$ and F_1-score$= 0.9208$ and the latter had much worse results, going from accuracy$= 0.7196$ and F_1-score$= 0.6509$ to accuracy$= 0.5132$ and F_1-score$= 0.4708$. It is interesting to observe that, as in sub-problem `L0`, the LS did not benefit from the use of the 'knn' kernel, having inferior results than LP, that uses 'rbf'. With DBSCAN the performance was similar but with a significant improvement in the LS results, that went from accuracy$= 0.8389$ and F_1-score$= 0.8001$ to accuracy$= 0.6889$ and F_1-score$= 0.6525$, and slightly worse results for the LP, going from accuracy$= 0.9722$ and F_1-score$= 0.9591$ to accuracy$= 0.9222$ and F_1-score$= 0.9152$. These results were obtained with `eps`$= 0.65$ and `min_samples`$= 2$, which corresponds to 62 clusters (including the noisy cluster) with maximum length of 12 labels.

It followed the scenario 3 approach, using again the K-means algorithm with 65 clusters, and the score obtained was 0.7460, which corresponds to a difference of more than 20% when compared to the best accuracy/F_1-score results obtained with the approach from scenario 2 and yet being better than the results of LS (regardless the amount of unlabeled data used).

This huge difference in the results, when compared to the sub-problems `L2` and `L3`, could be explained by a particularity that only happens in this sub-problem. When assessing the `L0` labels associated with the labels in each of the sub-problems, it was observed that the sub-problem `CAUSE` was the only one that had labels with more than one `L0` labels related to them. This happened essentially with the most common labels, e.g. “sem causa determinada”, that were usually linked to general texts that could be associated with all types of technical problems. Moreover, this was also the sub-problem with the highest number of features, since the set of all the combinations of the previous sub-problems labels is being included in the vocabulary. Another interesting characteristic of this sub-problem is that it has unique observations associated to each of the 3 `L0` labels, whereas the other sub-problems only have unique samples associated with one of them, ‘TV’ (although in `L2` and `L3` one of the unique observations corresponds to other `L0` label, ‘Voz’).

5.2 Discussion

The EM algorithm and the Self Learning model are both very fast although, in general, the EM algorithm was the model with the worst performance among all. This is a bit unfair, since the parameter α is intrinsically defined and therefore is always the same, which could not correspond to the best value. The Self Learning algorithm has better results than the EM algorithm, but usually worse than the rest of the models. Sometimes it has the same results as the baseline model, which is normal since it is based on the NB classifier and it is usually not affected by the unlabeled data (only in sub-problem `L1`).

It is difficult to define which model is better between LP and LS, since they do not always use the same kernel function. What could be observed is that the ‘knn’ kernel was typically associated with worse results, except in the sub-problem `L0`. Moreover, although it was expected that ‘knn’ would require less iterations and would be faster than ‘rbf’, this was not true in the majority of the cases, which was most likely related with the number of neighbors chosen and the amount of observations used being extremely high. Note that LS tries to correct some possible mistakes in the original labels by updating the labels of the known nodes to be consistent with its neighbors, while performing label propagation. This task is related to the α parameter and could also increase the computation time.

The dimensionality reduction techniques did not seem to have significant influence on the results and therefore were not further explored. The TF-IDF appeared to be the most appropriate text representation to use, due to its better performance in terms of results but also since it was the fastest method.

Contrary to what was expected, the use of the unlabeled data had, in most of the experiences, no effect or even a negative effect in the results. This could be due to the class imbalance (present in all sub-problems) but also due to the fact that the data does not satisfy the SSL assumptions described in Subsection 3.4.1. Although the class imbalance issue could not be successfully solved, since there was a significant number of unique observations that should not be removed (as a big portion of information would be lost and the results were actually worse), some of the SSL assumptions were forced to be satisfied in scenarios 2 and 3. In the sub-problems `L2`, `L3` and `CAUSE` it is clearly better to use the scenario 2 or 3 approaches than the scenario 1. In fact, for the sub-problem `CAUSE`, the best approach is the one from scenario 2, when using the LP.

Although the approach from scenario 3 was much faster, it was ignoring the majority of available data (the unlabeled set) and so the scenario 2 should be the implemented approach, despite the apparent irrelevance of the unlabeled data (caused by the class imbalance).

The results, in general, become worse as one goes further in the sub-problems, which could be related with the increase in the number of labels associated to them (that also increases the computation time) but also with the number of unique observations. Moreover, the dependence between the sub-problems is only represented by the words added to the texts, corresponding to the labels of the previous sub-problems, which might not be sufficient.

6

Conclusion

Contents

6.1 System Limitations and Conclusions

The main goal of this thesis, done in collaboration with Border Innovation, was to apply text mining and text categorization techniques to a set of troubleshoot guides from a telecommunications call center and use it in order to improve the recommendation system performed by its technical assistants. Since the data provided was mainly unlabeled, with a few labeled observations, the idea was to explore SSL models and how they could make use of all the available data.

This work was associated with several limitations, starting with the fact that the data was unstructured and most of the data observations were unlabeled. Moreover, as a lot of variables were involved in the process of diagnosis and troubleshooting of the client's problem, the analysis and the model construction had to be divided into 5 different sub-problems. In all of them, there was an evident class imbalance within the labeled set. Plus, since the text data was extracted by a web crawler that only fetched the webpages text elements (was not able to extract, for example, text included in images or diagrams), some parts were missing and therefore the quality of some texts was compromised.

This thesis starts with a review on the literature that addresses these topics, followed by an explanation of the background on the methods and techniques applied throughout the dissertation. All the steps of the text mining process are detailed, from text preprocessing to text representation (or transformation), dimensionality reduction and then text categorization and evaluation. Then the implementation of all these steps is described, including a previous data cleansing and exploratory analysis, along with the model configurations experimented. Finally, the results from each model setting are compared and discussed.

Although the results for the first two sub-problems were good (98% and 85% accuracy, respectively), the results for the following sub-problems in scenario 1 were quite bad. This way, it was not possible to achieve the full initial objective of constructing a new database, based on those results, that could be added to the recommendation system in order to improve the troubleshoot process performed by the technical assistant (indicating the correct troubleshoot guide to follow after all diagnosis steps were complete). Therefore, other approaches were explored (scenarios 2 and 3) with the aim of getting an alternative solution that could solve, at least partially, the initial problem. The alternative approach was to categorize the texts into a set of labels, smaller than the entire set, for each of the five sub-problems, starting by clustering the original set of labels and then classify the documents into the resulting clusters. The idea was to then provide these results to a specialist in order to help him with the diagnosis variables and troubleshoot guide association and thus saving time. This alternative solution was successfully achieved, obtaining more than 92% accuracy/F_1-score for the three sub-problems in which the texts were categorized into small sets of labels.

The unstructured data issue was solved with the preprocessing and transformation steps before text categorization, whereas the quality of the text data obtained by web crawling was not addressed. Even though the class imbalance could not be solved with the standard sampling techniques, this issue was partially overcome by including all the unique samples in the training set and then by clustering the labels. The approaches from scenario 2 and 3 also contributed to reorganize the data, imposing the SSL assumptions and thus obtaining great results with all the available data (including the unlabeled set).

Although, with the alternative scenario 2 approach applied to the last three sub-problems, the results obtained using the possible amount (up to 30%) of the unlabeled data were quite good, the effect of using the unlabeled data did not seem to be significant as it would be expected. This is related with the similarity of topics contained in each text, which makes the SSL assumptions difficult to satisfy, even if clustering the possible classes, as there is still the class imbalance problem that affects the quality of the clustering and the class separability (e.g. the same label being included in several clusters).

6.2 Future Work

Having the partial solution for the main goal of this work, there are still some possible developments that could be explored and implemented in order to enhance and complete this solution.

The first thing to do would be to implement the idea introduced in Subsection 4.5.2 in order to attribute the importance of each label in each cluster, suggesting the labels from the most to the least probable one and giving an extra help to the technical specialist that would do the association between the diagnosis variables and the troubleshoot guides.

Another important development would be to improve the quality of the texts used to train these model, by finding or developing a better web crawling method (e.g. using techniques that can extract text from images) but also by exploring the text preprocessing techniques for the Portuguese language and multilingual problems (since the texts are mostly written in Portuguese but have also some English expressions).

It could also be interesting to explore other SSL algorithms and implement them, since the solutions from `sklearn` (`LabelPropagation` and `LabelSpreading`) are very time consuming and need a lot of computer RAM when dealing with huge amounts of samples and, moreover, do not manage well with class imbalance.

After improving the data quality and the solution presentation, a specialist should use this in order to build the desired dataset to be added to the existing recommendation system and hence complete the solution, making the troubleshoot process of the call center more efficient and, in that way, improving it.

Bibliography

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *Proceedings of 31st Conference on Neural Information Processing Systems (NIPS 2017)*, 2017.

[2] "Center for computational and stochastic mathematics," http://cemat.ist.utl.pt/main.php, accessed: 2021-06-20.

[3] "Border innovation," https://border-innovation.com/, accessed: 2021-06-20.

[4] E. D. Liddy, "Natural language processing," *Encyclopedia of Library and Information Science*, 2001.

[5] D. S. McNamara, L. Allen, S. Crossley, M. Dascalu, and C. A. Perret, "Natural language processing and learning analytics," *Handbook of learning analytics*, pp. 93–104, 2017.

[6] D. Jurafsky and J. H. Martin, *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, 2020.

[7] V. B. Kobayashi, S. T. Mol, H. A. Berkers, G. Kismihok, and D. N. Den Hartog, "Text classification for organizational researchers: A tutorial," *Organizational research methods*, vol. 21, no. 3, pp. 766–799, 2018.

[8] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[9] M. Toman, R. Tesar, and K. Jezek, "Influence of word normalization on text classification," *Proceedings of International Symposium on Instrumentation System, Circuits and Transducers (INSCIT2006)*, vol. 4, pp. 354–358, 2006.

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[11] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in Neural Information Processing Systems*, 2013.

[12] J. E. Alvarez, “A review of word embedding and document similarity algorithms applied to academic text,” *Bachelor thesis, University of Freiburg*, 2017.

[13] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “Bert: Pre-training of deep bidirectional transformers for language understanding,” *arXiv preprint arXiv:1810.04805*, 2018.

[14] Y. Jernite, S. R. Bowman, and D. Sontag, “Discourse-based objectives for fast unsupervised sentence representation learning,” *arXiv preprint arXiv:1705.00557*, 2017.

[15] A. Rogers, O. Kovaleva, and A. Rumshisky, “A primer in bertology: What we know about how bert works,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 842–866, 2020.

[16] A. Ettinger, “What bert is not: Lessons from a new suite of psycholinguistic diagnostics for language models,” *Transactions of the Association for Computational Linguistics*, vol. 8, pp. 34–48, 2020.

[17] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, “Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter,” *arXiv preprint arXiv:1910.01108*, 2019.

[18] F. Souza, R. Nogueira, and R. Lotufo, “BERTimbau: pretrained BERT models for Brazilian Portuguese,” in *9th Brazilian Conference on Intelligent Systems, BRACIS, Rio Grande do Sul, Brazil, October 20-23 (to appear)*, 2020.

[19] Y. Yang and J. O. Pedersen, “A comparative study on feature selection in text categorization,” *Proceedings of International Conference on Machine Learning (ICML1997)*, vol. 97, no. 412-420, p. 35, 1997.

[20] J. R. Vergara and P. A. Estévez, “A review of feature selection methods based on mutual information,” *Neural computing and applications*, vol. 24, no. 1, pp. 175–186, 2014.

[21] R. A. Johnson and D. W. Wichern, *Applied multivariate statistical analysis*. Pearson London, UK, 2014, vol. 6, ch. 8.

[22] B. Rosario, “Latent semantic indexing: An overview,” *Techn. rep. INFOSYS*, vol. 240, 2000.

[23] D. M. Blei, A. Y. Ng, and M. I. Jordan, “Latent dirichlet allocation,” *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[24] Z. Liu, “High performance latent dirichlet allocation for text mining,” Ph.D. dissertation, Brunel University, School of Engineering and Design, 2013.

[25] G. Maskeri, S. Sarkar, and K. Heafield, “Mining business topics in source code using latent dirichlet allocation,” *Proceedings of the 1st India Software Engineering Conference*, pp. 113–120, 2008.

[26] A. Daud, J. Li, L. Zhou, and F. Muhammad, “Latent Dirichlet Allocation (LDA) and topic modeling: models, applications, future challenges, a survey,” *Frontiers of Computer Science in China*, vol. 4, 06 2010.

[27] W. M. Darling, “A theoretical and practical implementation tutorial on topic modeling and gibbs sampling,” *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies*, pp. 642–647, 2011.

[28] D. Christou, “Feature extraction using latent dirichlet allocation and neural networks: a case study on movie synopses,” *Bachelor thesis, University of Macedonia*, 2016.

[29] T. Norsten, “Exploring the potential of twitter data and natural language processing techniques to understand the usage of parks in Stockholm,” *KTH, Royal Institute of Technology*, 2020.

[30] M. Zareapoor and K. Seeja, “Feature extraction or feature selection for text classification: A case study on phishing email detection,” *International Journal of Information Engineering and Electronic Business*, vol. 7, no. 2, p. 60, 2015.

[31] B. F. Ljungberg, “Dimensionality reduction for bag-of-words models: PCA vs LSA,” *Standford University*, 2017.

[32] D. Berrar, “Bayes’ theorem and naive bayes classifier,” *Encyclopedia of Bioinformatics and Computational Biology: ABC of Bioinformatics; Elsevier Science Publisher: Amsterdam, The Netherlands*, pp. 403–412, 2018.

[33] H. Zhang, “Exploring conditions for the optimality of naive bayes,” *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 19, no. 02, pp. 183–198, 2005.

[34] A. Likas, N. Vlassis, and J. J. Verbeek, “The global k-means clustering algorithm,” *Pattern recognition*, vol. 36, no. 2, pp. 451–461, 2003.

[35] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, “A density-based algorithm for discovering clusters in large spatial databases with noise.” *Proceedings of 2nd Internationl Conference on Knowledge Discovery and Data Mining (KDD96)*, vol. 96, no. 34, pp. 226–231, 1996.

[36] A. Huang *et al.*, “Similarity measures for text document clustering,” *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, vol. 4, pp. 9–56, 2008.

[37] A. Amine, Z. Elberrichi, and M. Simonet, “Evaluation of text clustering methods using wordnet.” *Int. Arab J. Inf. Technol.*, vol. 7, no. 4, pp. 349–357, 2010.

[38] W. Usino, A. S. Prabuwono, K. Allehaibi, A. Bramantoro, A. Hasniaty, and W. Amaldi, “Document similarity detection using k-means and cosine distance,” *International Journal of Advanced Computer Science and Applications*, vol. 10, 02 2019.

[39] S. Dang, “Performance evaluation of clustering algorithm using different datasets,” *International Journal of Advance Research in Computer Science and Management Studies (IJARCSMS2015)*, vol. 3, pp. 167–173, 2015.

[40] C.-H. Lee and H.-C. Yang, “Construction of supervised and unsupervised learning systems for multilingual text categorization,” *Expert Systems with Applications*, vol. 36, no. 2, 2009.

[41] Y. Ko and J. Seo, “Automatic text categorization by unsupervised learning,” *Proceedings of The 18th International Conference on Computational Linguistics (COLING2000)*, vol. 1, 2000.

[42] O. Chapelle, B. Schölkopf, and A. Zien, *Semi-Supervised Learning*. The MIT Press, 2006.

[43] I. Triguero, S. García, and F. Herrera, “Self-labeled techniques for semi-supervised learning: taxonomy, software and empirical study,” *Knowledge and Information systems*, vol. 42, no. 2, pp. 245–284, 2015.

[44] A. P. Dempster, N. M. Laird, and D. B. Rubin, “Maximum likelihood from incomplete data via the em algorithm,” *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 39, no. 1, pp. 1–22, 1977.

[45] J. Bagherzadeh and H. Asil, “A review of various semi-supervised learning models with a deep learning and memory approach,” *Iran Journal of Computer Science*, vol. 2, no. 2, pp. 65–80, 2019.

[46] A. Bocancea, “Supervised classification leveraging refined unlabeled data,” *Master thesis, Linköping University*, 2015.

[47] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[48] M. Hossin and M. Sulaiman, “on evaluation metrics for data classification evaluations,” *International Journal of Data Mining & Knowledge Management Process*, vol. 5, no. 2, p. 1, 2015.

[49] D. D. Lewis, “Evaluating and optimizing autonomous text classification systems,” *Proceedings of the 18th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 246–254, 1995.

[50] P. J. Rousseeuw, “Silhouettes: a graphical aid to the interpretation and validation of cluster analysis,” *Journal of computational and applied mathematics*, vol. 20, pp. 53–65, 1987.

[51] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo, “A comparison of extrinsic clustering evaluation metrics based on formal constraints,” *Information retrieval*, vol. 12, no. 4, pp. 461–486, 2009.

[52] Y. Xu and R. Goodacre, “On splitting training and validation set: A comparative study of cross-validation, bootstrap and systematic sampling for estimating the generalization performance of supervised learning,” *Journal of Analysis and Testing*, vol. 2, no. 3, pp. 249–262, 2018.

[53] D. Berrar, “Cross-validation,” *Encyclopedia of bioinformatics and computational biology*, vol. 1, pp. 542–545, 2019.

[54] M. Bekkar, H. K. Djemaa, and T. A. Alitouche, “Evaluation measures for models assessment over imbalanced data sets,” *J Inf Eng Appl*, vol. 3, no. 10, 2013.

[55] P. Branco, L. Torgo, and R. P. Ribeiro, “Relevance-based evaluation metrics for multi-class imbalanced domains,” *Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 698–710, 2017.

[56] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, “Smote: synthetic minority over-sampling technique,” *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[57] N. V. Chawla, N. Japkowicz, and A. Kotcz, “Special issue on learning from imbalanced data sets,” *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[58] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, “The imbalanced training sample problem: Under or over sampling?” *Proceedings of Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*, pp. 806–814, 2004.

[59] J. W. Osborne and A. Overbay, “The power of outliers (and why researchers should always check for them),” *Practical Assessment, Research, and Evaluation*, vol. 9, no. 1, p. 6, 2004.

[60] R. Feldman, J. Sanger *et al.*, *The text mining handbook: advanced approaches in analyzing unstructured data*. Cambridge University Press, 2007.

[61] G. G. Chowdhury, “Natural language processing,” *Proceedings of Annual review of information science and technology*, vol. 37, no. 1, pp. 51–89, 2003.

[62] A. C. B. Forte, “Análise de comentários de clientes com o auxílio a técnicas de text mining para determinar o nível de (in)satisfação,” *tese de Mestrado, Universidade do Porto*, 2015.

[63] R. Garside, S. Fligelstone, and S. Botley, “Discourse annotation: Anaphoric relations in corpora in Garside R., Leech G. and McEnery A.(eds) Corpus Annotation: Linguistic information from computer text corpora,” Addison Wesley Longman, London, 1997.

[64] S. C. S. Pinto, “Processamento de linguagem natural e extração de conhecimento,” Ph.D. dissertation, Universidade de Coimbra, 2015.

[65] K. Dave, S. Lawrence, and D. M. Pennock, “Mining the peanut gallery: Opinion extraction and semantic classification of product reviews,” *Proceedings of the 12th international conference on World Wide Web*, pp. 519–528, 2003.

[66] S. García and F. Herrera, “Evolutionary training set selection to optimize c4. 5 in imbalanced problems,” *Proceedings of Eighth International Conference on Hybrid Intelligent Systems*, pp. 567–572, 2008.

[67] M. Kusner, Y. Sun, N. Kolkin, and K. Weinberger, “From word embeddings to document distances,” *Proceedings of International conference on machine learning*, pp. 957–966, 2015.

[68] S. Pan and T. Ding, “Social media-based user embedding: A literature review,” *Proceedings of the twenty-eight International Joint Conference on Artificial Intelligence (IJCAI19)*, 2019.

[69] M. U. Gutmann and A. Hyvärinen, “Noise-contrastive estimation of unnormalized statistical models, with applications to natural image statistics.” *Journal of Machine Learning Research*, vol. 13, no. 2, 2012.

[70] L. Chen, “Curse of dimensionality in LIU, LING and ÖZSU, M. TAMER.(eds) Encyclopedia of Database Systems,” pp. 545–546, 2009.

[71] D. Kalman, “A singularly valuable decomposition: the svd of a matrix,” *The college mathematics journal*, vol. 27, no. 1, pp. 2–23, 1996.

[72] S. T. Dumais, G. W. Furnas, T. K. Landauer, S. Deerwester, and R. Harshman, “Using latent semantic analysis to improve access to textual information,” *Proceedings of the SIGCHI conference on Human factors in computing systems*, pp. 281–285, 1988.

[73] P. Tan, M. Steinbach, A. Karpatne, and V. Kumar, *Introduction to Data Mining*. Pearson Publication, London, 2019, ch. 3.

[74] L. Rokach and O. Maimon, “Clustering methods,” in *Data mining and knowledge discovery handbook*. Springer, 2005, pp. 321–352.

[75] S. Ghoche, “Real-time tf-idf clustering using simhash, approximate nearest neighbors, and dbscan,” Ph.D. dissertation, Harvard University, 2016.

[76] E. Schubert, J. Sander, M. Ester, H. P. Kriegel, and X. Xu, “DBSCAN, revisited: why and how you should (still) use DBSCAN,” *ACM Transactions on Database Systems (TODS)*, vol. 42, no. 3, pp. 1–21, 2017.

[77] X. Zhu and Z. Ghahramani, “Towards semi-supervised classification with markov random fields,” *Carnegie Mellon University*, 2002.

[78] Y. Bengio, O. Delalleau, and N. Le Roux, *Label propagation and quadratic criterion*. The MIT Press, 2006, ch. 11.

[79] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf, “Learning with local and global consistency,” in *Advances in neural information processing systems*, 2004, pp. 321–328.

[80] I. Guyon, K. Bennett, G. Cawley, H. J. Escalante, S. Escalera, T. K. Ho, N. Macià, B. Ray, M. Saeed, A. Statnikov, and E. Viegas, “Design of the 2015 chalearn automl challenge,” *Proceedings of 2015 International Joint Conference on Neural Networks (IJCNN)*, 2015.

[81] “Python software foundation, python language reference, version 3.6.9.” https://www.python.org/, accessed: 2021-09-02.

[82] “Google research, “google colaboratory”,” https://colab.research.google.com/notebooks/intro.ipynb, accessed: 2021-09-02.

[83] W. McKinney *et al.*, “Data structures for statistical computing in python,” *Proceedings of the 9th Python in Science Conference*, vol. 445, pp. 51–56, 2010.

[84] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in python,” *Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[85] D. Naber, “Integrated tools for spelling, style, and grammar checking,” *Proceedings of OpenOffice.org Conference, Barcelona*, 2007.

[86] M. Honnibal and I. Montani, “spacy 2: Natural language understanding with bloom embeddings,” *(to Appear)*, 2017.

[87] “Acordo ortográfico - portal da língua portuguesa,” http://www.portaldalinguaportuguesa.org/acordo.php, accessed: 2021-05-26.

A

Exploratory Data Analysis

This appendix contains some of the graphics that were obtained during the exploratory data analysis. The following figures describe the data in terms of absolute frequency of the classes for each sub-problem, as well as the absolute frequency of the possible text lengths.

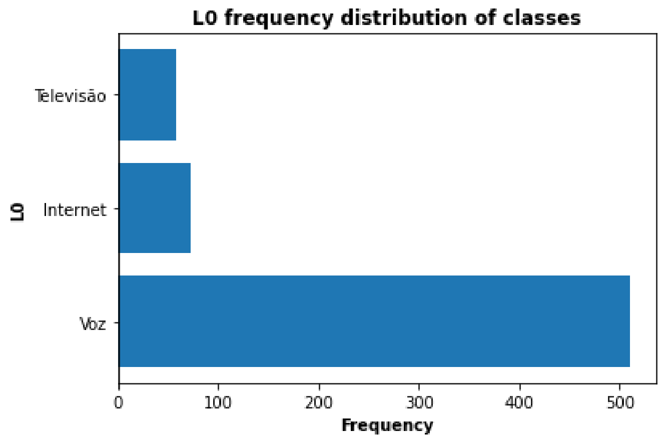

Figure A.1: Absolute frequency distribution of the classes for sub-problem L0.

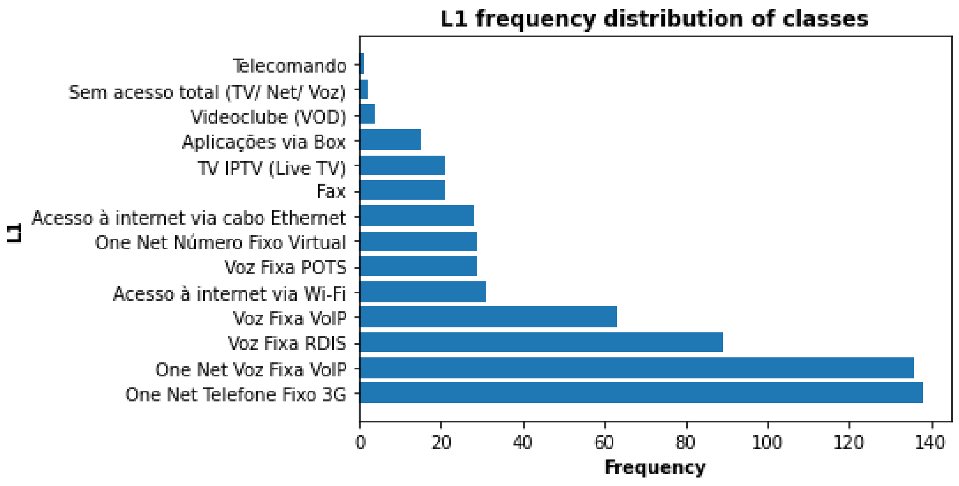

Figure A.2: Absolute frequency distribution of the classes for sub-problem L1.

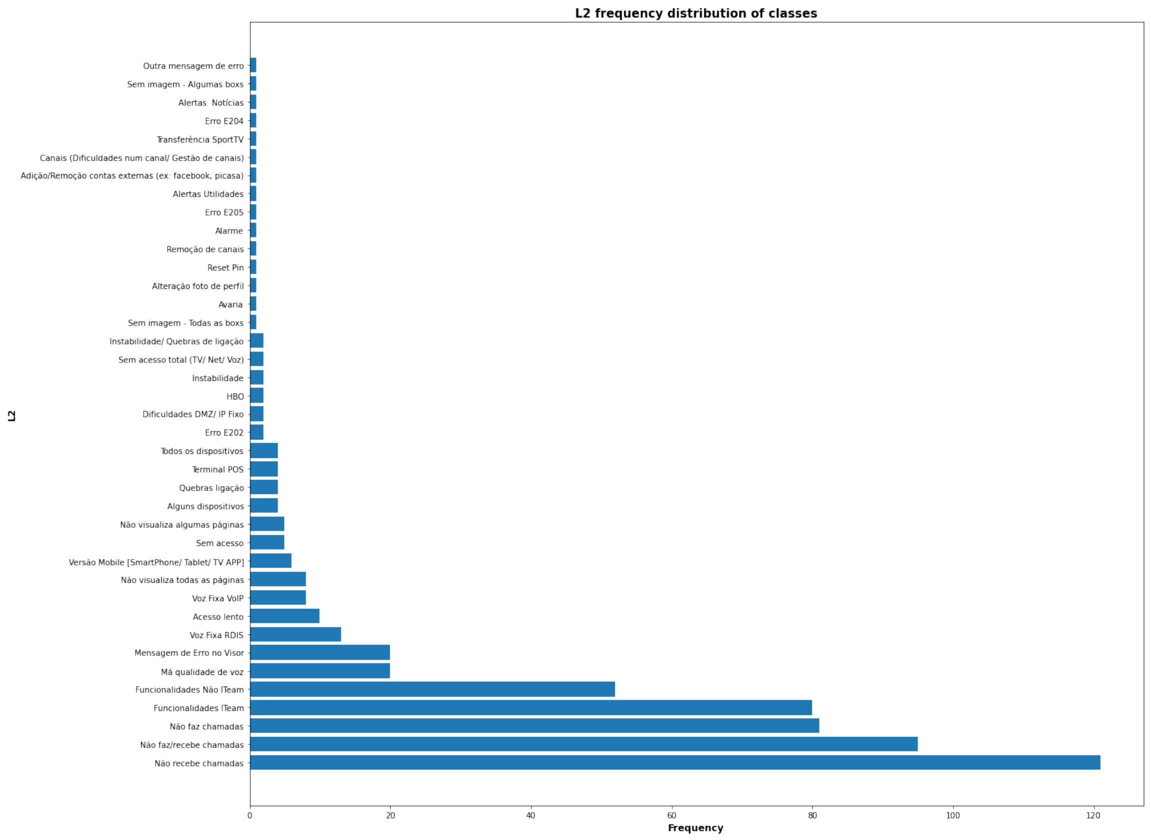

Figure A.3: Absolute frequency distribution of the classes for sub-problem L2.

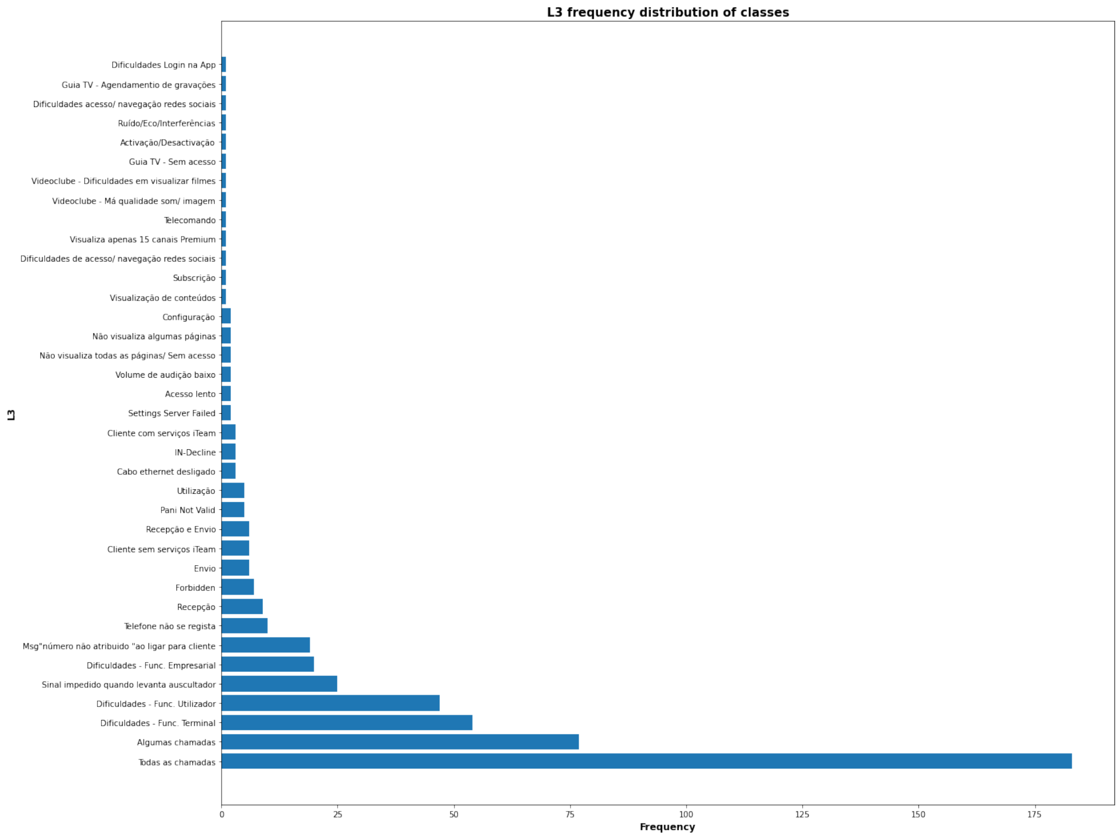

Figure A.4: Absolute frequency distribution of the classes for sub-problem L3.

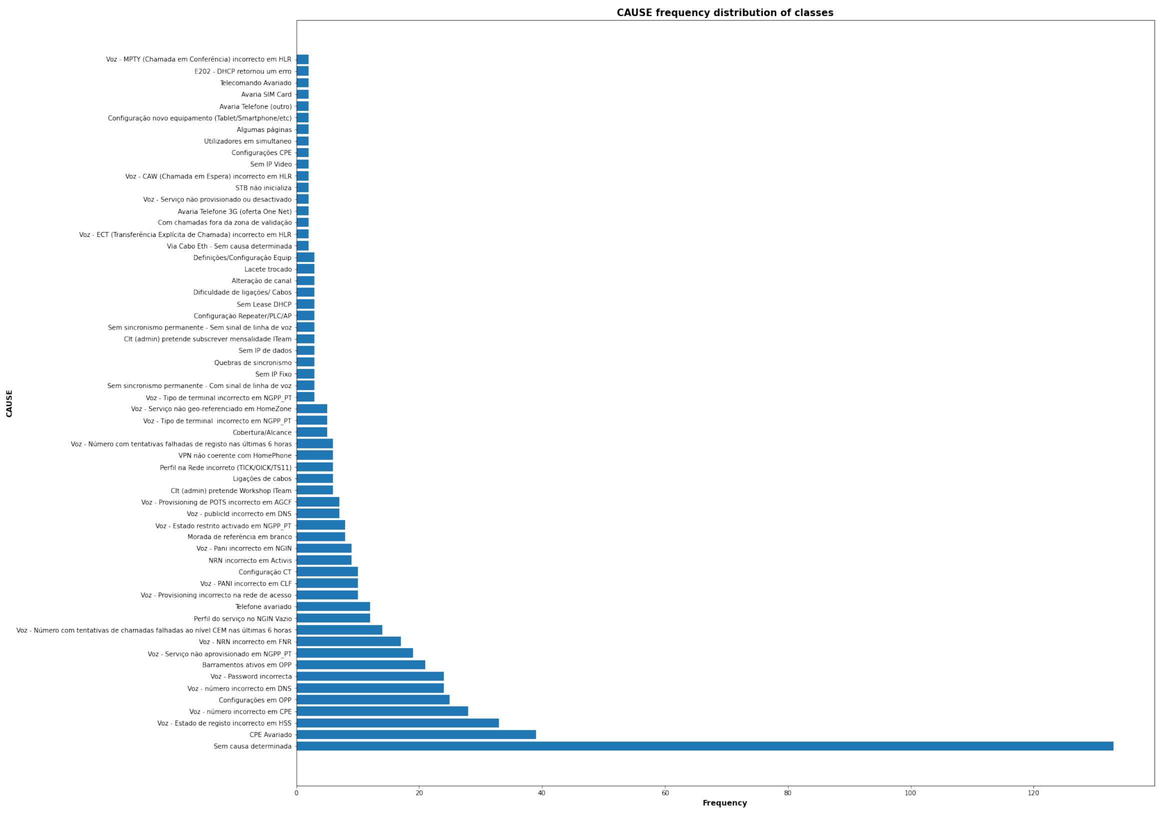

Figure A.5: Absolute frequency distribution of the classes for sub-problem CAUSE.

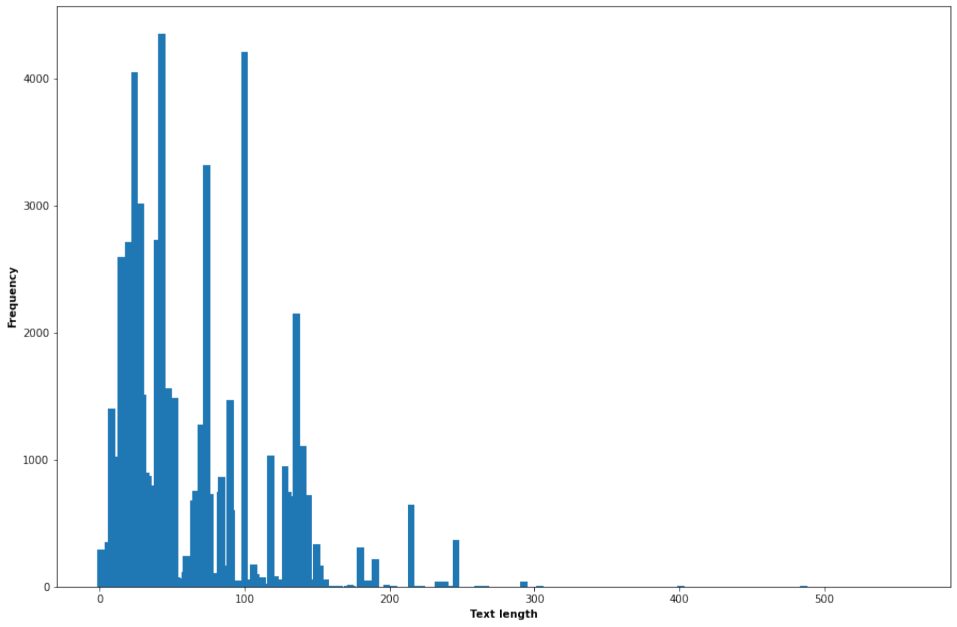

Figure A.6: Absolute frequency distribution of text lengths.

B Results

In this appendix, the results of part of the performed experiences are presented in tables, including the models applied, the evaluation metrics used and other aspects of the experience (e.g. different text representation methods and/or dimensionality reduction techniques).

Table B.1: Results for L0 with TF, Word2Vec and Doc2Vec text representations.

Metric	*% unlabeled data*	**Naive Bayes**			**Label Propagation**			**Label Spreading**		
		TF	**W2V**	**D2V**	**TF**	**W2V**	**D2V**	**TF**	**W2V**	**D2V**
Accuracy	0	0.9741	0.8756	0.8912	0.8756	0.7979	0.7979	0.9326	0.8394	0.8497
	10				0.8394	0.7979	0.7979	0.8912	0.8497	0.8135
	30				0.8342	0.7979	0.7979	0.8756	0.8394	0.8497
F1-score	0	0.9732	0.8911	0.8810	0.8515	0.7082	0.7082	0.9283	0.8502	0.8175
	10				0.7924	0.7082	0.7082	0.8925	0.8647	0.8272
	30				0.7836	0.7082	0.7082	0.8828	0.8570	0.8648

Table B.2: Results for `L1` with TF-IDF, unique observations included after the train/test split and dimensionality reduction techniques.

Model	*Metric*	*% unlabeled data*	**Chi2**	**Mutual Information**	**PCA**	**LSA**	**LDA**
Label Propagation	**Accuracy**	0	0.8077	0.8077	0.8022	0.7747	0.7967
		10	0.6484	0.6484	0.6429	0.6538	0.6374
		30	0.6319	0.6319	0.6319	0.6319	0.6264
	F1-score	0	0.7888	0.7888	0.8068	0.7761	0.7818
		10	0.6791	0.6791	0.6608	0.6927	0.6777
		30	0.6477	0.6477	0.6588	0.6477	0.6683
Label Spreading	**Accuracy**	0	0.8462	0.8462	0.8462	0.8462	0.6593
		10	0.8352	0.8352	0.8242	0.8242	0.7253
		30	0.8352	0.8352	0.8242	0.8242	0.7143
	F1-score	0	0.8315	0.8315	0.8317	0.8320	0.6171
		10	0.8209	0.8209	0.8188	0.8188	0.6909
		30	0.8210	0.8209	0.8148	0.8148	0.6623
Naive Bayes $\alpha = 0$	**Accuracy**	0	0.8352	0.8352	0.4890	0.4780	0.6374
	F1-score		0.8158	0.8158	0.4014	0.3629	0.5768

Table B.3: Results for `L1` with TF, unique observations included after the train/test split and dimensionality reduction techniques.

Model	*Metric*	*% unlabeled data*	**None**	**Mutual Information**	**LSA**
Label Propagation	**Accuracy**	0	0.8022	0.8022	0.8022
		10	0.6538	0.6484	0.6374
		30	0.6648	0.6484	0.6429
	F1-score	0	0.7861	0.7861	0.7862
		10	0.6916	0.6875	0.6548
		30	0.6972	0.6769	0.6598
Label Spreading	**Accuracy**	0	0.6538	0.6538	0.6484
		10	0.4176	0.4176	0.4176
		30	0.3516	0.3352	0.3132
	F1-score	0	0.5740	0.5740	0.5728
		10	0.3514	0.3514	0.3514
		30	0.2447	0.2307	0.2084
Naive Bayes $\alpha = 0$	**Accuracy**	0	0.8407	0.8407	0.7308
	F1-score		0.8240	0.8240	0.7421

Table B.4: Results for `L1` with Word2Vec, unique observations included after the train/test split and dimensionality reduction techniques.

Model	*Metric*	*% unlabeled data*	**None**	**Mutual Information**	**LSA**
Label Propagation	**Accuracy**	0	0.7637	0.7857	0.7527
		10	0.6813	0.6813	0.4505
	F1-score	0	0.7460	0.7740	0.7394
		10	0.6794	0.6872	0.4911
Label Spreading	**Accuracy**	0	0.3626	0.3901	0.4011
		10	0.2253	0.2253	0.2253
	F1-score	0	0.2939	0.2663	0.3336
		10	0.0828	0.0828	0.0828
Naive Bayes $\alpha = 0$	**Accuracy**	0	0.7473	0.7747	0.6374
	F1-score		0.7608	0.7510	0.6654

Table B.5: Results for `L1` with Doc2Vec and unique observations included after the train/test split.

Metric	*% unlabeled data*	**Naive Bayes**	**Label Propagation**	**Label Spreading**
Accuracy	0	0.7363	0.7308	0.2253
	10		0.5989	0.2253
	30		0.5989	0.2253
F1-score	0	0.7464	0.7041	0.0828
	10		0.6392	0.0828
	30		0.6392	0.0828