View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Trepo - Institutional Repository of Tampere University

Tampere University

Honain Mohib Derrar

CLUSTERING FOR THE AUTOMATIC ANNOTATION OF CUSTOMER SERVICE CHAT MESSAGES

Faculty of Computing and Electrical Engineering
Master of Science Thesis
January 2019

ABSTRACT

Honain Mohib Derrar: Clustering for the automatic annotation of customer service chat messages
Master of Science Thesis
Tampere University
Master's Degree Program in Information Technology
January 2019

The objective of this thesis work is to identify a clustering setting that provides human annotators with the support they need to perform topic annotation of customer service chat data.

Many of the customer service chat automation tools available involve the use of supervised machine learning techniques to learn how to answer a customer query based on historical conversations between a customer and a customer service agent. While this approach has provided satisfying results for many use cases, it still represents a challenge since annotation work incurs large costs.

In order to alleviate some of the challenges faced by the annotation team at ultimate.ai, we seek to provide a solution using clustering approaches that helps reduce the annotation workload by providing as many correct chat message annotations as possible automatically. At the same time, the approach needs to be easily usable and applicable to chat data from different languages and industries while not requiring immense computational resources.

The approach used in this work improves upon the previously used clustering baseline in the company and identifies a clustering evaluation metric that enables further internal research to continuously improve the clustering of customer service chat data. Finally, metric learning is explored in a effort to improve the obtained results even further.

Keywords: Clustering, Machine Learning, Natural Language Processing, Customer Service

The originality of this thesis has been checked using the Turnitin OriginalityCheck service.

PREFACE

This thesis work concludes one of the most exciting chapters of my life. The years I spent at Tampere University of Technology (now Tampere University) allowed me to grow from the academic perspective as well as the human perspective. The experiences I lived and the friends I made are an integral part of the person I have become.

The realization of this thesis wouldn't have been possible without the support of several people. First of all, I would like to express my deepest gratitude to Pr. Heikki Huttunen for accepting to examine this thesis work and for all the guidance and learning opportunities he has provided me with.

I would also like to thank Pr. Rohit Babbar for his useful comments and recommendations during the writing of the present document.

This work was done at ultimate.ai. I would like to thank all of my colleagues in the company who always supported me and pushed me to do my best. More particularly, I would like to thank Jaakko Pasanen for always challenging my ideas and making me think outside of the box. I would also like to extend my gratitude to all the members of our customer success team (annotation team): Joonas Suoranta, Sara Ekholm and Pekka Stenlund who were always there to answer my questions and to evaluate some of my experimental results.

In addition, I would like to thank my friends who were always there for me. More particularly, my friends here in Finland who always knew how to cheer me up.

I would like to express my love and gratitude to my parents who taught me the values of learning and hard work and who have always supported me and been there for me. Finally, I would like to thank my sister for always believing in me and being my first supporter.

Helsinki, 19th January 2019

Honain Mohib Derrar

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
AMI	Adjusted Mutual Information
ANN	Artificial Neural Network
ARI	Adjusted Rand Index
CBOW	Continuous Bag Of Words
CFG	Context Free Grammar
CNN	Convolutional Neural Network
DBCV	Density-Based Clustering Validation
DBMT	Direct-Based Machine Translation
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
HDBSCAN	Hierarchical Density-Based Spatial Clustering of Applications with Noise
IDF	Inverse Document Frequency
LSTM	Long Short-Term Memory
MI	Mutual Information
MLP	MultiLayer Perceptron
MMC	Mahalanobis Metric for Clustering
MST	Minimum Spanning Tree
MT	Machine Translation
NLI	Natural Language Inference
NLP	Natural Language Processing
NLTK	Natural Language ToolKit
PCA	Principal Component Analysis
PCFG	Probabilistic Context Free Grammar
ReLU	Rectified Linear Unit
RI	Rand Index
SC	Silhouette Coefficient
SG	Skip Gram
SNLI	Stanford Natural Language Inference
TF	Term Frequency

1 GENERAL INTRODUCTION

Customer service is one of the most challenging jobs in today's economy. Indeed, in addition to a good product, a company needs to deliver a quality customer experience in order to keep current customers happy and to attract new customers.

With the development of the internet, more and more companies are delivering their customer service experience online and reducing the resources dedicated to more mature platforms such as phone-based customer service. Online, customers can ask questions to customer service agents through chat. One of the main advantages of chat, from the company's perspective, is that one agent can handle multiple requests at the same time which is not possible with more traditional phone-based systems. This means less costs for the companies but at the same time more stress and less work satisfaction for the customer service agents.

In order to improve the work satisfaction of customer service agents and help reduce costs for companies even further, ultimate.ai, a Finnish start-up, aims at helping boost the productivity of a business's customer service unit. The way ultimate.ai does this, is by providing companies with tools for them to deliver better, faster responses to their customers queries and to reduce the burden of complex and repetitive tasks.

In order to deliver on its promise, ultimate.ai makes use of machine learning technologies to offer tools that companies use to serve customers in a better way. The machine learning approaches currently used in the company are mostly using supervised machine learning techniques i.e machine learning techniques that require human annotated data.

Human annotations are obtained by dedicating a team of people who go through a company's historical chat data, usually containing tens of thousands of messages, while labelling customer messages according to the topic they contain, for instance, greeting or product return. As one can clearly imagine, the annotation process takes a lot of time and human resources thus, making the on-boarding of ultimate.ai's customers slower and, potentially, limit the scalability of our tools.

In this work, we deal with the issue of supporting the annotation work by providing a solution that annotates as many customer messages as possible in an automatic fashion thus, significantly reducing the workload of the annotation team and, at the same time, helping ultimate.ai bring value to our customers faster. Overall, our objectives are to provide tools that improve upon the current approaches currently in-use at the company while making sure that our developed approach is applicable to data coming from different languages and industries. In addition, our approach should be easily usable by people with little knowledge of machine learning.

This thesis is organized as follows. Chapter 2 and 3 cover topics related to Natural Language Processing (NLP) and Machine Learning. Specifically, Chapter 2 introduces basic concepts related to the automatic processing of natural language and outlines some of the challenges facing NLP practitioners while Chapter 3 covers concepts related to Machine Learning and gives examples on the way it could be applied to NLP. Chapter 4 presents the problem we are trying to solve in detail and introduces the important concepts that are used to tackle our problematic. Chapter 5 builds upon the previous chapters and describes the methodology of our approach, the data we used for our experiments, the results we obtained and a discussion of our findings. Finally, Chapter 6 contains concluding remarks and potential research directions for future works.

2 NATURAL LANGUAGE PROCESSING

2.1 Introduction

Given that chat conversations are carried out using natural language, we will dedicate this chapter to the question of Natural Language Processing (NLP). Language is probably one of the most intriguing skills that humankind was able to develop as it allows us to express our minds [1], communicate with others and, more significantly, learn. It is present in virtually all aspects of our lives to the point where most of us would not imagine living without being able to use it and understand it in one way or another.

Although humans are generally able to understand and manipulate language effortlessly, challenges may still arise. According to [25], in 2018 almost 13 Million text messages have been sent and approximately 176 000 Skype calls were made across the globe every single minute. These numbers show the extraordinary amount of textual and spoken language data available in the world today, and they are likely to grow in the future. This means that we cannot rely solely on human expertise to understand and take advantage of linguistic data anymore; automation is now a necessity.

In addition to volume, other challenges pertaining to the manipulation of natural language data arise. For example, the translation of texts between different languages to ensure that knowledge is accessible to the biggest number of people makes the analysis, understanding and manipulation of linguistic data a key challenge for humankind. Another common linguistic challenge relates to the fact that, in certain places (like in the Arab world), spoken language can be largely different depending on the context where it is used thus, making a thorough analysis of the language challenging.

In this chapter, we will discuss Natural Language Processing (NLP) i.e the use of computers to solve language related problems. We will cover some of its current applications and discuss few of the challenges facing NLP researchers and practitioners.

2.2 What is Natural Language Processing?

Natural Language Processing is a field at the intersection of computer science, artificial intelligence and linguistics that aims at exploring ways to use computers for analyzing, understanding and manipulating natural language data (human language), be it text or speech [21]. In other words, NLP is about creating and manipulating systems and algorithms that take as input or produce as an ouptut natural language data [36].

As language is a fundamental part of human communication, NLP applications can be found almost everywhere in our daily lives. For instance, the spell-checkers in our text editors or the auto-complete features of our mobile phone keyboards are manipulating natural language data. They make our lives easier and have probably started changing our attitudes towards language, as it is not rare to see people mistype words intentionally for convenience (smaller distance for typing on a keyboard) because they know that their phone's keyboard will take care of fixing their mistakes for them. A study [58], shows that the types of errors made in student papers in the United States have changed between the 1980s to 2008 with spelling mistakes and wrong word usage moving to higher positions in the list of mistakes. Although technology might not be the sole reason for this change, it is likely that NLP systems in general have had an impact on the way we manipulate and interact with language. These changes show that natural language is not still and keeps on changing and evolving with time and external influences hence, posing multiple challenges to researchers and practitioners.

As language is used to communicate ideas and information, ways to analyze the contents of a piece of text or speech need to be devised. According to [32, 56], meaning is extracted from natural language (written or spoken) based on 7 interdependent levels:

1. *Phonetic (phonological) level* that deals with the way language is pronounced. It is important for spoken language,
2. *Morphological level* that focuses on morphemes i.e the smallest part of a word that carries meaning (prefixes, suffixes...),
3. *Lexical level* that focuses on the study and analysis of the language's lexicon i.e an inventory of the words in the language,
4. *Syntactic level* that focuses on the grammatical structure of sentences from different aspects,
5. *Semantic level* that focuses on the meaning of words and their disambiguation,
6. *Discourse level* that focuses on the meaning and structure of larger texts i.e not only on a sentence level,
7. *Pragmatic level* that incorporates knowledge from the world into the interpretation of a given document,

One given NLP application does not have to focus on all levels. It is worth noting that, typically, the higher the level, the more difficult the application becomes as knowledge that is not directly present in a given text or speech is often needed for accurate processing. Accordingly, depending on the level they require, some NLP tasks have been mostly solved, others require more research efforts to be solved.

2.3 Applications of Natural Language Processing

Natural Language Processing applications can be very diverse ranging form spell checking to machine translation and question-answering. In what follows, we will review some Natural Language Processing applications starting by the simplest applications to the hardest.

2.3.1 Spell Checking

Spell checking is a task aiming at detecting and correcting spelling mistakes in texts and can be used alone or in conjunction with other linguistic tools [38]. It is present in most text editing software nowadays.

Spelling errors stem from different sources, the most important being [78]:

1. *Typographical errors* which are the result of a user mistyping a word. This type of errors is usually predictable as it is mostly related to wrong hand gestures on a keyboard leading to a misuse of neighbouring keys.
2. *Author ignorance errors* which are the result of the writer's ignorance of a word's correct spelling. These errors usually are the result of a difference between a word's pronunciation and its spelling.
3. *Storage and transmission errors* which are mostly related to the mechanisms through which texts are stored and transmitted.

Although some spelling correction techniques have been devised specifically for a particular type of error, most spell checking systems perform error detection followed by error correction.

One technique used for error detection is called dictionary look-up [89]. The idea is that, given a list of candidate words within a text of interest, we check whether each word can be found in a dictionary of correctly spelled words. If the word cannot be found, it is flagged (detected) as a spelling error. Once an incorrectly spelled word has been detected, its closest words in the dictionary can be suggested to the user. One of the similarity criteria used for determining the most similar words is called *edit distance* [54].

Edit distance is defined as the smallest number of editing operations needed to change a word into another, with editing operations being insertion, deletion and substitution [54]. The notion of *edit distance* can also be used in other contexts, for instance, to measure similarity between two DNA sequences.

Depending on the costs associated with each operation, the similarity results can be different. Figure 2.1 shows an example for finding the edit distance between words "intention" and "execution" [49].

From Figure 2.1, we can see that, to transition from 'Intention' to 'Execution', we need to

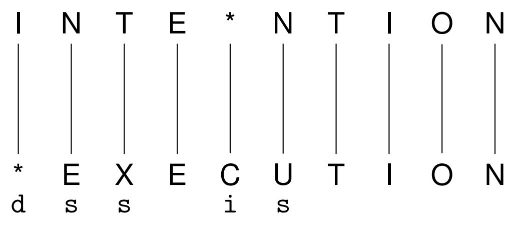

***Figure 2.1.** Operations needed to change "Intention" into "Execution"*

perform, at least, 1 deletion, 1 insertion and 3 substitutions. If we assign a cost of 1 to each operation, we will obtain an edit distance of 5. Other cost models exist where, for instance, substitutions have a cost of 2 in which case the edit distance would become 8. The most similar word from the edit distance perspective would be the one with the smallest overall cost.

Spell checking has been around for decades now and many of the tools publicly available nowadays are efficient and can detect complex errors hence, the choice to have spell checking as the first (less complicated) NLP task we cover.

2.3.2 Syntactic Parsing

The term "parsing" originally refers to the grammatical explication of sentences [72]. In applications of Natural Language Processing, the term refers to the syntactic analysis of sentences. The notion of parsing has often been linked to the notion of grammars in formal languages. Although parsing can be performed using formal grammars i.e production rules describing how sentences are created in our language, it is not a necessity. Current trends in the syntactic parsing literature such as in [18, 36, 84] show that data-driven approaches are the leading approaches nowadays. These two types of approaches (grammar-driven and data-driven) represent the two main approaches to syntactic parsing [72].

In grammar-driven parsing, a formal grammar G is used as a definition of our language $L(G)$, say, English. Figure 2.2 [72] contains an example of a Probabilistic Context-Free Grammar (PCFG) which are an extension of the Context-Free Grammars (CFG), probably the most widely used type of formal grammar which was pioneered by the work of Chomsky. The Figure contains a grammar for the sentence "Economic news had little effect on financial markets." where, in addition to the rules of production, we have probabilities of using each rule hence, the term probabilistic. The symbols used are based on the Penn Treebank's syntactic tag-set where, for instance, NP refers to Noun Phrase, PU to Punctuation and JJ to Adjective [88].

Conversely, in data-driven approaches, the syntactic parsing of a sentence is usually produced based on models learned with the use of corpora of syntactically annotated sentence structures called Treebanks [22]. One of the most iconic and used Treebanks is the Penn Treebank [62].

S → *NP VP PU*	1.0	*JJ* → *Economic*	0.3
VP → *VP PP*	0.3	*JJ* → *little*	0.5
VP → *VBD NP*	0.7	*JJ* → *financial*	0.2
NP → *NP PP*	0.2	*NN* → *news*	0.4
NP → *JJ NN*	0.5	*NN* → *effect*	0.6
NP → *JJ NNS*	0.3	*NNS* → *markets*	1.0
PP → *IN NP*	1.0	*VBD* → *had*	1.0
PU → *.*	1.0	*IN* → *on*	1.0

Figure 2.2. *Probabilistic context-free grammar for an English sentence*

Both the grammar-driven and the data-driven approaches are used for producing syntactic representations. Even though there are different types of syntactic representations, the representation based on the notion of constituents has been dominant during the last 50 years [72]. In this type of representation, a given sentence is decomposed into segments named constituents. Such constituents are categorized in a way that reflects their internal structure.

Figure 2.3 contains the constituency tree of the sentence that can be produced using the grammar introduced in Figure 2.2. As we can see, we have a sentence root splitting into different components (constituents) which recursively split into other components. The leaves of the tree are represented by the words of the sentence. It is worth noting that the tags used in the figure also follow the Penn Treebank notation.

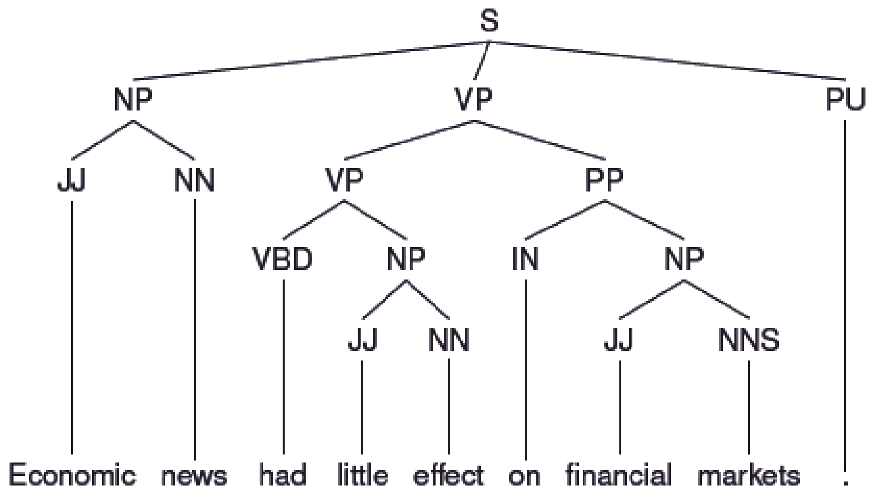

Figure 2.3. *Constituent structure for an English sentence.*

As we have seen from our brief review, syntactic parsing is not a trivial task since any endeavour towards achieving good syntactic representations requires some level of linguistic knowledge for understanding the structure of a language which is difficult to automate. In addition to that, syntactic parsing can be used in downstream tasks such as question-answering which makes it a potentially crucial component in any endeavour towards improving other Natural Language Processing tasks.

2.3.3 Machine Translation

According to [42] more than 7000 languages are currently spoken around the globe. While this language diversity is beautiful in theory, practice shows that there are a limited number of dominant languages such as English, Spanish or Chinese. This creates a situation where users of smaller languages are forced to learn a dominant language to have access to knowledge thus, putting them in a disadvantageous situation compared to speakers of more powerful languages. Moreover, in the context of a globalized economy, tourists, workers and students are more likely to travel to areas in which the official language is foreign to them thus, having the tools that help people understand and communicate thoughts in different languages is vital.

According to [74], two main approaches exist for Machine Translation: single approach methods and hybrid approach methods. Single approach methods, as their name indicates, only make use of one method for performing a translation while hybrid approach methods combine statistical approaches with other approaches including those used in single approach systems.

Some examples of the single approach methods include the Direct-Based Machine Translation (DBMT) which is one of the most basic approaches to Machine Translation as it just replaces words from the source language with words in the target language without performing linguistic analysis. Instead, a bilingual dictionary is used for finding the corresponding words in the target language [74].

As for the hybrid approach methods, we can mention the word-based models [14] which are alignment models as they model lexical dependencies between individual words before performing the translation.

Nowadays, many of the approaches used in Machine Translation involve the use Machine Learning methods where parallel corpora, in other words, corpora of texts accompanied with their respective translation in the target language are used like in [6, 59].

Machine Translation, while it is still a work in progress as it can be seen from results given by some of the most popular translation software, can potentially help improve the quality of other NLP tasks by allowing to augment the amount of data available for languages with small data-sets. Compared to, for instance, syntactic parsing, it can be noticed that it is more complex since, among other things, it works on 2 languages simultaneously. However, a lot of promising developments have been achieved over the past few years which indicate that further key developments are likely to happen in the future.

2.4 Challenges in Natural Language Processing

After covering some Natural Language Processing applications, we will try to cover some of the reasons that make processing language in an automatic way difficult. Indeed, even if NLP methods have seen a tremendous development in the last decades [8], processing Natural Language data in an automatic way is still a challenging task for many reasons.

One of the sources of complexity relates to the fact that human language is highly ambiguous [36], for instance, the phrase *I saw a man on a hill with a telescope* [90] could have several interpretations like:

- A man is on a hill, he has a telescope and I saw him.
- A man is on a hill, I used a telescope to see him.

As humans, we typically use our knowledge of the world and context we are in to infer the correct interpretation for a sentence. For example, we could understand that the speaker used a telescope if our previous discussion with them was about a new telescope they got. However, machines rarely have access to the knowledge we have as humans.

Non-verbal communication is yet another issue that faces the NLP community as, in many cases, spoken or written language lacks certain aspects of communication like body language which is an important part of human communication as many insights on a person's emotion can be drawn from it [26].

Another challenge facing the NLP practitioners lies in the fact that language is symbolic, compositional and sparse [36]. Indeed, the basic building blocks of human language i.e characters and words are categorical elements. This means that there is no simple mathematical operation that would allow us to transition from the word *happy* to the word *sad* . In contrast, we could give the example of color in image processing where it is fairly easy to transition from a color image to a gray-scale image as color is a continuous notion.

Language is compositional meaning that grouping characters produces meaning that goes beyond the meaning of individual characters. Similarly grouping words produces meaning that goes beyond the meaning of the individual words in a sentence. This is particularly visible in idiomatic expressions whose meaning cannot be derived from the meaning of its individual components (words). One such example of this would be the English phrase "going on a wild goose chase" which typically does not mean chasing the animal but rather making a meaningless endeavour.

The combination of the symbolic and compositional properties leads to the issue of data sparseness: we can combine words in an almost infinite number of ways to produce meaning. This means that, in most cases, NLP systems (and in many cases humans) will only see a finite number of possible combinations thus, no matter how many linguistic examples we possess, we are likely to face unseen events in real-life applications.

Another type of challenge arises from the domain from which linguistic data is extracted. Indeed, processing data from chat messages raises different challenges then processing data coming from, say, scientific papers. One notable difference is higher number of spelling mistakes in chat data compared to scientific papers.

Finally, we will mention the issues of language representation and feature extraction. Indeed, given that NLP systems are about processing linguistic data with the use of computers, there needs to be a transformation from natural language to machine language i.e numbers. Depending on the way we represent our linguistic data, there might be more emphasis on some level of language meaning: morphology, semantics etc. Furthermore, knowing how to represent language for the task at hand is not sufficient as we also need to decide what type of information we need to represent. For instance, we could represent individual words or pairs of successive words in a given sentence.

2.5 Conclusion

This chapter gave us the opportunity to introduce concepts related to Natural Language Processing. We first gave some definitions of NLP then, we performed a small review of few applications of Natural Language for which, in many cases, the scientific community is currently focusing extensively on data-driven approaches involving the use of machine learning techniques. Finally, we concluded the chapter by a coverage of some of the challenges that make Natural Language Processing difficult despite recent developments in the field. In the next chapter we will cover basic machine learning concepts and review their relationship with Natural Language Processing.

3 MACHINE LEARNING FOR NATURAL LANGUAGE PROCESSING

3.1 Introduction

As we have seen in the previous chapter, a lot of Natural Language Processing tasks are not trivial to model hence, many of the popular approaches nowadays make use of Machine Learning techniques together with corpora of linguistic resources in order to infer a satisfactory model for the task at hand. In this chapter, we will cover basic concepts related to Machine Learning and see how they can be applied in the context of Natural Language Processing.

3.2 What is Machine Learning?

The field of machine learning sparks a lot of interest among individuals as well as among the business community. This can clearly be seen through the skyrocketing number of online courses covering the topic and the number of companies claiming to use it.

As we pointed out previously, a great amount of linguistic data that requires automated processing is available today. This trend is not only visible with linguistic data but also with a lot of other types of data such as images from social media or geo-localization information from our hand-held devices. Being able to understand and extract knowledge from big amounts of data can potentially be beneficial for humans and the environment. Unfortunately, knowledge about a given field of application and about the nature of the data being processed is not sufficient for making sense of the vast volumes of content available today. Algorithms that automate the study of data become fundamental.

Machine Learning is a domain of artificial intelligence whose aim is to make machines exhibit an intelligent behaviour similar to that of humans by making them learn from their environment [29]. This is typically done by feeding the machines with data regarding the task they need to perform. Machine Learning algorithms have been applied, with a certain degree of success, to different disciplines such as computer vision [51], health-care [19] or fraud detection [95].

A more formal definition of Machine Learning was given by [68] which states that Machine

Learning includes any computer program whose performance improves at some tasks with experience. More specifically, a computer program is considered to learn through experience E for tasks T using a performance metric P if its performance at tasks T improves with experience E.

Tasks typically involve processing data samples that have been measured or prepared using domain knowledge in order for the system to perform a specific operation on them like recognizing the category of a data sample. Performance measures, on the other hand, try to quantify the learning progression using suitable indicators for the task at hand, for instance, mean-squared error.

As for the experiments, they depend a lot on the structure of the data used. 3 of the major variants of machine learning techniques can be seen from Figure 3.1 [20]. The figure explains what type of Machine Learning approaches are used depending on the type of data available. When using human annotated data, the approaches are called supervised. Conversely, if our system only makes use of unannotated data, it is said to be unsupervised. Finally, if human annotated data is used together with unannotated data, the approach is considered to be semi-supervised.

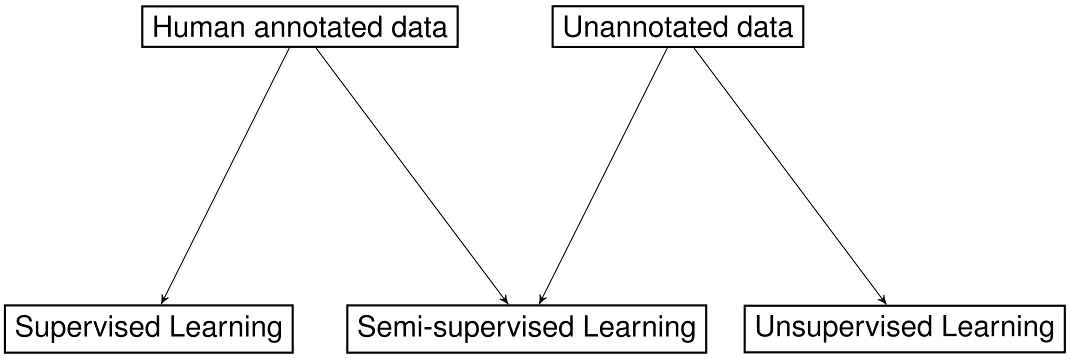

Figure 3.1. *Different types of learning based on the type of data used*

3.2.1 Supervised Machine Learning

Supervised Machine Learning systems learn to perform a task using human-labelled examples for the given task [28]. The term Human is important here as it means that the data labelling process went through human scrutiny thus, ensuring a higher quality of labelling. Many of the approaches using Machine Learning today make use, with some level of success, of supervised techniques as they are usually fast to train and yield satisfactory results. Unfortunately, obtaining human annotations is costly. Indeed, human labor is typically expensive both in terms of time and financial resources. Not to mention the disagreement issues where human experts do not agree on annotations to give to certain data samples.

Formally, the goal of supervised machine learning is to learn a mapping between inputs $\mathbf{x}$ and targets $\mathbf{y}$ using a dataset of annotated pairs $(\mathbf{x_i}, \mathbf{y_i})$ with $\mathbf{x_i} \in \mathbf{X}$ the input space and $\mathbf{y_i} \in \mathbf{Y}$ the target space with the constraint that the $(\mathbf{x_i}, \mathbf{y_i})$ pairs are sampled independently and identically distributed form a distribution ranging over $\mathbf{X} \times \mathbf{Y}$ [17]. The main point here is to be able, from annotated examples, to generalize to new unseen examples.

One example of a supervised Machine Learning task would be sentiment analysis such as in [11]. In this type of task we would typically have (text, sentiment) pairs. This means that every piece of text in the training data would have the correct sentiment, like positive, negative or neutral, associated with it. The role of the system would be to be able to predict the sentiment of new non-annotated texts after learning from the corpus of annotated data.

Evaluating the learning process

As mentioned earlier, a system is said to be learning when it is improving a performance metric through experience. At the same time, we said that the essence of supervised learning was the ability to generalize from a set of human-annotated examples to new, unseen, examples.

One typical approach to evaluating the generalization ability of supervised approaches is to split the data into 3 sets: the *training set* and two held-out sets called *validation set* and *test set* respectively (typically, the training set is larger than the other sets). All tests, analyses and selection of the different models should be performed using the validation set. In other words, we train different variants of our system on the training set and select the best performing one based on a performance metric measured on the development set. Then, once the final model has been selected, a performance evaluation is run on the test set in order to obtain a meaningful estimate of the generalization ability of the model. The idea here is to keep the test set out of the experiments and only use it once we found our best approach using the development set thus, simulating a real-life situation where we will use our model with unseen examples [36].

3.2.2 Unsupervised Machine Learning

Unsupervised Machine Learning systems make use of unannotated data and try to discover patterns in it then, group elements with similar patterns together in a way that elements within a certain group are more similar to each other than to members of the other groups [28]. In a formal way, unsupervised learning aims at estimating a common distribution $\mathbf{X}$ using a set of n examples $(\mathbf{x_1}, \mathbf{x_2}, ..., \mathbf{x_n})$ with $\mathbf{x_i} \in \mathbf{X}$. Again here, the constraint is that the examples are drawn independently and identically distributed from the common distribution $\mathbf{X}$ [17].

The main advantage of unsupervised approaches is that they allow us to leverage the huge amounts of unannotated data available nowadays at moderate cost as no human annotations are required. In addition, unsupervised methods are considered by some renowned researchers to be the future of Artificial Intelligence (AI). Unfortunately, unsupervised approaches are still not providing good enough results in many cases. Indeed, making patterns visible and deciding what patterns to give priority to is not a trivial task as even humans might not agree on what is similar and what is not. In addition, unsupervised learning typically deals with volumes of data that are larger than the volumes that are dealt with in the supervised case thus, requiring larger computational capabilities.

Similarly to the supervised case, sentiment analysis can be solved from an unsupervised perspective like in [44]. In fact, many of the machine learning problems could, in theory, be approached from an unsupervised perspective in which case, the burden of human annotation would be reduced. It is worth noting that, in the context of this work, we will be mostly focusing on unsupervised approaches specifically because we cannot afford to have humans annotate data at a large scale.

3.2.3 Semi-Supervised Machine Learning

Semi-supervised Machine Learning approaches are a combination of supervised and unsupervised approaches. Indeed, a semi-supervised system would make use of a certain amount of human-annotated data together with a large amount of unannotated data thus, trying to make the most of both approaches [17]. Indeed, supervised approaches have proven their ability at handling many problems successfully and, as we mentioned previously, unsupervised approaches are considered to be the future of Artificial Intelligence.

One of the earliest approaches to semi-supervised learning is probably self-learning also called decision-directed learning. The idea of this approach is to start training using the initial set of labelled data then, using the learned decision function at each step, predicting labels for part of the unannotated data. The newly labelled (the system's own predictions) data is then added to the annotated part of the training data and the process is repeated several times [17].

As with the unsupervised case, the sentiment analysis task (as well as many other tasks) we mentioned earlier can also be handled using semi-supervised approaches [85].

3.3 Machine Learning Techniques: Neural Networks

After covering some of the different approaches to machine learning from the perspective of the type of data used, we will now to give a concrete example of a machine learning technique, namely, Neural Networks which, for simplicity, will be discussed from the perspective of supervised learning.

Artificial Neural Networks (ANN), commonly called Neural Networks, have initially been inspired by the brain's internal wiring which is made of computational components named neurons [36].

Artificial neurons take as input and produce as outputs scalar values. Each of the input values are associated with a specific weight. The inputs of an artificial neuron are multiplied with their respective weights then, the multiplication results are typically summed (other operations could be applied) and fed to a non-linear function which finally passes the result to the output of the neuron.

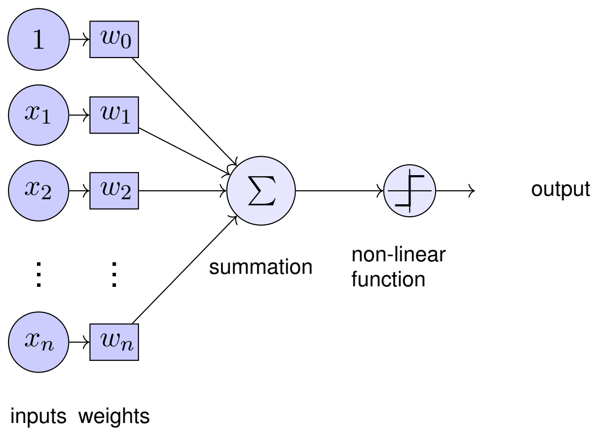

Figure 3.2. *Structure of an artificial neuron*

Figure 3.2 shows the typical structure of an artificial neuron. We can clearly see the inputs of the neuron denoted as x_i which are associated with weights w_i. We can see that there is a weight w_0 associated with an input with a value of 1; the weight is called a *bias* and represents the response of the neuron when the inputs to it are all 0. As described earlier, once the multiplication between the inputs and the weights is performed, a summation is applied followed by a non-linearity thus, producing the output of the artificial unit.

When several artificial neurons are connected together, we obtain an Artificial Neural Network. Such networks have been used to approximate complex functions such as in [51]. In fact, it has been shown that, if weights are learned correctly, a neural network with the right activation function (non-linear function) and a sufficient number of neurons can approximate various mathematical functions [24, 41].

Figure 3.3 contains the example of a feed-forward neural network structure (feed-forward means that the connections between the different neurons do not form a cycle and that data in the network moves forward) where several artificial neurons are connected. Neurons are arranged in layers whose number can be varied. In addition, the number of neurons per layer can also be adjusted. The flow of information passes from inputs to outputs passing through the hidden layers. It is worth noting that each arrow after the input layer represents the weights associated with the inputs of the unit in question and, that the input of the different neurons are vectors having as many dimensions as the number of arrows they receive. For instance, neurons in the hidden layer of Figure 3.3 would have 4-dimensional inputs and the output layer neuron would have a 5-dimensional in-

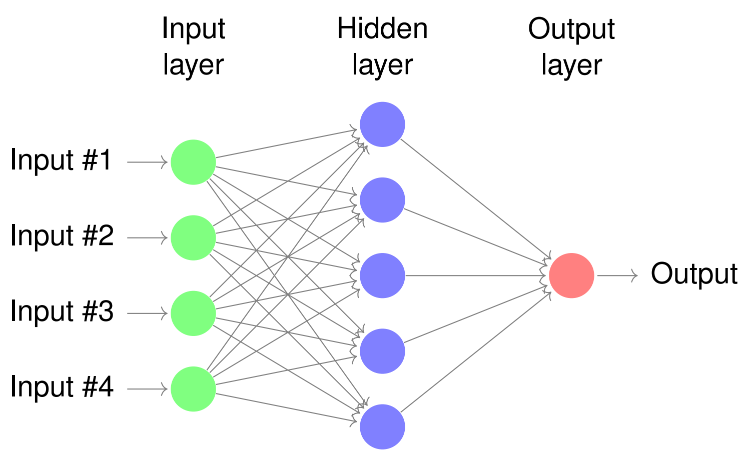

Figure 3.3. *Structure of a feed-forward Neural Network*

put. In essence, the flow of information in a neural network is just a series of matrix multiplication.

In their simplest form, feed-forward neural networks can be composed of only one input and one output layer (such structure is called a perceptron and is just a linear model) thus having the following mathematical formulation in which $\mathbf{x}$ is the input vector, $\mathbf{W}$ represents the weight matrix and b is a bias term:

$$Perceptron(x) = \mathbf{Wx} + \mathbf{b}$$

Where $\mathbf{x} \in \mathbb{R}^{d_{in}}, \mathbf{W} \in \mathbb{R}^{d_{out} \times d_{in}}$ and $\mathbf{b} \in \mathbb{R}^{d_{out}}$. When (hidden) layers are added to the perceptron, we obtain a MultiLayer Perceptron (MLP). Depending on the number of hidden layers and their number of units, more complex functions can be expressed. In Figure 3.3, we have a MLP with one hidden layer. Its mathematical formulation is the following:

$$MLP(x) = \mathbf{W_2}g(\mathbf{W_1x} + \mathbf{b_1}) + \mathbf{b_2}$$

Where $\mathbf{x} \in \mathbb{R}^{d_{in}}, \mathbf{W_1} \in \mathbb{R}^{d_1 \times d_{in}}, \mathbf{b_1} \in \mathbb{R}^{d_1}, \mathbf{W_2} \in \mathbb{R}^{d_2 \times d_1}$ and $\mathbf{b_2} \in \mathbb{R}^{d_2}$. Here, $\mathbf{W_1}$ and $\mathbf{b_1}$ represent the weight matrix and bias for the input's first linear transformation while g represents the activation function (a non-linear function) of the hidden layer's neurons. As for $\mathbf{W_2}$ and $\mathbf{b_2}$, they represent the weight matrix and bias for transforming the hidden representation of the data to the output. It is worth noting that adding layers to an MLP would follow the same mathematical pattern.

3.3.1 Activation Functions

So far, we have been mentioning the activation or non-linear functions without providing examples of their nature. The choice of the activation function is mostly related to the task at hand and there is no agreed upon criteria for choosing one at the moment [36]. There are various alternatives to activation functions but one of the most popular activation functions is the *Sigmoid function* also known as *logistic function*. Its formulation is the following:

$$\sigma(x) = \frac{1}{(1+e^{-x})}$$

Another popular activation function is the *Rectifier (ReLU)* [35]. Units (neurons) using the rectifier function are called Rectified Linear Units. This activation function is easy to compute and offers the benefit of not having the issue of yielding a near 0 gradient like for the sigmoid activation which has vanishing gradients when its values are near 1 for instance. This makes ReLU units particularly suited for a setting with multiple layers. The Rectifier function is defined as follows:

$$ReLU(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases} \tag{3.1}$$

3.3.2 Training of Neural Networks

So far, we have been covering the way artificial neurons and neural networks work and are structured. However, in order to be able to approximate a function $f()$ between inputs $\mathbf{x}$ and targets $\mathbf{y}$, the network needs to learn the correct mapping between $\mathbf{x}$ and $\mathbf{y}$. In essence, this comes down to finding the correct set of weights $\mathbf{W}$ and biases $\mathbf{b}$ in the different units [36].

Let's assume we are dealing with a supervised task where we are given input vectors $\mathbf{x_{1:n}}$ and their corresponding labels $\mathbf{y_{1:n}}$. In order to evaluate the quality of our predictions $\mathbf{\hat{y}_{1:n}}$ compared to the desired targets $\mathbf{y_{1:n}}$, a *loss function* $L(\mathbf{\hat{y}}, \mathbf{y})$ is defined where $\mathbf{\hat{y}} = f(\mathbf{x})$ with $f()$ being the learned approximation of our mapping. The function L returns a scalar value that quantifies the loss suffered when the predicted value is $\mathbf{\hat{y}}$ while the true output should have been $\mathbf{y}$ for an (input, target) pair.

Building on this, we can say that, using a set of (input, target) pairs $(\mathbf{x_{1:n}}, \mathbf{y_{1:n}})$, a loss function L and a parameterized function $f(\mathbf{x}; \Theta)$, we can define a loss value for all the training data for parameters Θ (the weights and biases of our network) as follows:

$$\mathcal{L}(\Theta) = \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x_i}; \Theta), \mathbf{y_i})$$

Based on this, finding the set of parameters Θ yielding the minimum loss and, by extension, providing the best approximation for our desired mapping is equivalent to finding Θ for which $\mathcal{L}$ is minimized as follows:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}(\Theta) = \underset{\Theta}{\operatorname{argmin}}\, \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x_i}; \Theta), \mathbf{y_i})$$

The previous equation is typically non-convex and is usually solved using gradient-based approaches such as *Stochastic Gradient Descent*. It is worth noting that the minimization of the loss might lead to over-fitting i.e the model will be scoring very well on the training data but will not be able to generalize to new, unseen data. To combat this issue, some restrictions are typically added so that the learned parameters are not too complex thus, avoiding over-fitting. The updated equation to minimize becomes the following:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}}\, \mathcal{L}(\Theta) = \underset{\Theta}{\operatorname{argmin}}\, \frac{1}{n} \sum_{i=1}^{n} L(f(\mathbf{x_i}; \Theta), \mathbf{y_i}) + \lambda R(\Theta)$$

Where R is a *regularization term* taking the parameters Θ and returning a scalar value reflecting their "complexity" for which different notions apply depending on the regularization term used. For instance some terms penalize larger weights. The value λ is used to specify the importance of the regularization term.

3.4 Using Natural Language with Machine Learning

Until now, we have covered general aspects of different types of Machine Learning approaches and introduced one popular Machine Learning technique. In this section, we will focus on the specific usage of linguistic data in Machine Learning systems, more specifically, textual data. Indeed, we mentioned earlier that we feed the different systems with data, however, we did not cover the form in which it should be fed. This is a fundamental step in any effort to use Machine Learning approaches.

Before being able to feed language to our system, it needs to be transformed into fixed-size real-valued vectors that contain the relevant pieces of information for the system to optimally perform since, in general, Machine Learning systems require that we tell them what type of information to look at in order for them to perform well. This process is called *feature extraction*. The nature of features we extract and the way they are extracted is based on the task to be performed and is a determining aspect in the success or failure of our different endeavours [36]. One aspect worth noting here is the potential challenge of representing data into a fixed-sized vector when linguistic data can vary in length and presents the challenges that we outlined in the previous chapter such as, ambiguousity, contextuality and compositionality.

Before determining the type of features one needs to extract, it is important to define the type of problem at hand. We will briefly discuss some types of NLP problems from the perspective of the entities they deal with [36]:

- **Word level problems:** this type of NLP problems deals with words (any character sequence separated by white spaces) and tries to answer questions regarding, for instance, the word type (Verb, Adjective, ...).

- **Text level problems:** deal with linguistic data spanning multiple words and tries, for example, to determine the content of the text like "greeting" or "weather query".
- **Paired-text level problems:** handle problems where we need to answer questions regarding pairs of words or texts and cover topics like identifying whether pairs of texts are translations of each other

In the preceding points, we didn't cover all existing typologies of problems as it is not the focus of this document. The main point here is that, different problems might require different feature extraction approaches and different machine learning methods in order to succeed as, unlike humans, Machine Learning systems might not be able to notice the features alone. For instance, tasks focusing on texts will probably require information on the order of words and their relationships. In addition, the typology of the problem will define what type of annotations we have when using a supervised machine learning approach as, for instance, word-level problems will most likely need word level annotations. Conversely, a text level problem will typically require annotated chunks of texts and not necessarily of words when dealt with using a supervised Machine Learning approach.

We can give the example of features for sentence classification. Sentence classification aims at assigning a label (class) to an input sentence. For instance, the sentiment analysis task we mentioned during this chapter, is a sentence classification task.

In this type of problematic, we can consider the counts of the words as features [36]. For instance, a positive sentence will probably contain words like "happy" or "wonderful" while a negative sentence would probably contain terms such as "bad" or "horrible". Of course, sequences of words called *n-grams* (a sequence of 2 words is 2-grams, a sequence of 3 words is 3-gram etc) can be informative as well.

Based on this, one could compile the vocabulary V that is used in all the sentences to process and represent a sentence by vector v of dimension $|V|$ where each v_i would contain the frequency of the word having index i in the sentence at hand. Such a representation, is called a *bag of words*. The word bag here reflects the fact the word order of the words in the sentence is not preserved in this representation. We will cover this type of approach in more detail in the next chapter.

Of course one could think of adding more features such a bag of n-grams, where instead of only counting the number of occurrences of individual words, we would count the occurrences of n-grams in the different sentences in order to provide information about word order to our system. The main point here is focus on extracting relevant features to make the training process simpler for the Machine Learning algorithm while keeping in mind the computational requirements of training. For example, in the case of the bag of words, one sentence could end up being represented by a huge vector if we have a large $|V|$ (and we often do) thus, limiting the usability of the bag of n-grams as there are many ways of combining two words. This can be linked to the data sparsity issue that was discussed earlier and will usually mean that the sparser the representation the more complex architectures are needed for training which, in turn, means more training data requirements.

One technique to reduce the complexity of the task would be to pre-process the words in all the sentences at hand and get their roots thus, making words like "heavy" and "heavily" have the same form which, in many use-cases, is sufficient and allows to reduce the size of $|V|$. This technique is called stemming. To remove an additional layer of complexity for the machine learning system, one could normalize all the words in sentence to use lowercase characters to prevent words like "WORD" and "word" to be seen by the system as different as they would have different entries in a bag-of-words representation. Finally, another typical way of making sure that machine learning systems have only access to the relevant parts of a text is stop-word removal. Stop-words are words are usually the most common words of a language such as "The", "a" or "to". They rarely provide valuable information to the Machine Learning system and can usually be removed [71].

Overall, 2 issues need to be identified when applying machine learning techniques on linguistic data, namely, what features to extract (number of words, etc) in order to strike the right balance between the expression potential of the features and the computational burden they represent and how to represent the extracted features (bag-of-words or other technique).

3.5 Conclusion

In this chapter we have covered general principles of Machine Learning and introduced a formal definition for it. We also covered different types of approaches based on the type of data available. In addition, we introduced neural networks, a very popular machine learning technique. We concluded the chapter by discussing the issue of features in the context of Natural Language Processing. These concepts are going to be useful in the next chapters as we discuss the problem we are trying to solve and the way we plan to solve it.

4 CLUSTERING AND CUSTOMER SERVICE CHAT DATA

4.1 Introduction

Until now, we have covered theoretical aspects related to Natural Language Processing and Machine Learning in the context of Natural language processing. In this chapter, we will discuss the problem we are trying to solve for this thesis work, why it is important and what type of approaches we can use to solve it. Furthermore, we will discuss some previous works related to our problematic.

4.2 ultimate.ai: Customer Service Automation

This work is done for ultimate.ai, a Finnish start-up founded in 2016. The company aims at providing customer service agents within companies with the tools they need to provide better, faster responses to customer queries. Indeed, customer service work is challenging as it often requires answering several repetitive requests every day (often simultaneously) as fast as possible. This usually decreases the quality of the service, meaning potential losses for the company, and puts strain on the agents thus, reducing their work satisfaction. As a result, ultimate.ai works on developing tools that will, ultimately, improve the way customer service is handled across different industries.

Several tools are being developed in the company, all of them involving processing textual natural language data and, some of them making use of Machine Learning approaches. One of the main tools currently developed is called *Suggestion Engine*. This tool provides customer service agents with a small number of reply suggestions based on a customer's text message thus, reducing their work strain as they do not have to search for an answer and type it, instead, they just click on the most relevant suggestion or, in few cases, add a custom reply that will later be used for training and improving the engine so that it gives more relevant suggestions. For instance, if a customer asks about the way they can reset a password, then, based on the question, the *Suggestion Engine* would suggest, say, 4 possible replies ranked from the most confident to the least confident. This system has been shown to significantly improve the productivity of customer service agents with several of our clients.

Another product being developed at ultimate.ai is called *Automation Engine*. In this case, there is no human monitoring the system as it automatically shows the customer (person asking the question) the answer that it thinks is the most relevant given the problem at hand.

The main point here is that the two systems we just described provide recommendations based on our clients' historical data i.e we analyze data from previous text conversations between customers and customer service agents then, we annotate a subset of the customer messages based on their content while prioritizing the customer queries that we deem as important. The annotated data can then be used to train one of the supervised learning systems we described above. For instance, reset password messages could be tagged <reset_password> meaning that all chat messages relating to this issue would have the same annotation like:

- I need help to reset my password = <reset_password>,
- How can I reset my password = <reset_password>,
- What should I do to reset my password = <reset_password>,

When in use, the system will predict <reset_password> which is associated with the desired reply. For instance: "to reset your password you may click on the reset password link".

Of course, one needs to keep in mind that the examples we just gave are not necessarily representative of all historical messages. Indeed, given that chat messages are highly conversational, they can significantly vary in length and in style. For example, some customers like to put their question into one message. Others would split it into several messages. Furthermore, some customers might have different issues and write all their questions into one given message thus, having a situation where one chat message would require multiple annotations. Another issue that makes our task harder relates to the number of typos and the use of spoken language in chat messages.

4.3 The problem: Unsupervised Message Categorization

As we have just seen, the company's current Machine Learning-based solutions make use of supervised approaches to achieve the desired goal: providing customer service agents with the required tools to perform their work better and faster.

While this approach has delivered good results across multiple industries, it still represents a bottleneck when it comes to on-boarding new clients. Indeed, the amount of historical data that needs to be analyzed and annotated is huge as it sometimes contains hundreds of thousands or millions of chat messages not to mention issues related to the quality of these messages thus, taking a lot of time and human resources to properly annotate data. Moreover, the annotators need to have some level of knowledge about the industry at hand if they are to produce quality annotations. These factors lead to higher costs for our clients especially because of the need to recruit employees who understand the data and are able to annotate it correctly and because of the time it takes to produce a usable set of annotated customer messages.

Given the aforementioned challenges, the problem we need to solve becomes: *How to automatically perform the annotation work in order to reduce the workload of human annotators?* More specifically, we want to develop a solution that will provide the annotation team with tools that extract as many clean groups of messages as possible automatically. A clean group of messages is a group that contains messages that are mostly about the same topic such as reset password. At the same time, we would like to avoid having a large number of message groups referring to the same topic i.e we would like to put most messages from a given category into the same group and not split them across multiple groups. Finally, given the variable quality of messages in our historical data, we expect messages that are not clear or that deal with multiple topics at the same time (messages that are hard to deal with) to be grouped together in a sort of "others" category. A system with such specifications would allow us to obtain a starting data-set for training our supervised systems fairly quickly and provide insights regarding the content of historical messages and in turn value to our customers.

In addition to the aforementioned description of our desired system, there is a set of constraints that we are faced with while trying to achieve this goal. Their aim is to make the end solution as practical as possible and easily usable by people with little knowledge of Machine Learning and Natural Language Processing. The most important constraints are:

- The approach used needs to be easily applicable across different languages: this means that using features, techniques or additional data only available for one or a subset of the languages we work with is not an option. The solution needs to be as language agnostic as possible.
- The approach needs to be easily applicable to data from different industries: this means that using features that are specific to one given industry should not be considered.
- The approach should scale: this means that we should be able to process tens of thousands of messages simultaneously using reasonably priced computational resources as this cost will be directly transferred to our customers.
- Results should be provided in a reasonable amount of time: This means that our approach should return results quickly, if possible.

The two first constraints relate to the fact that new customers from different industries and, to some extent, working with different languages should be on-boarded using the proposed solution at a reasonable (small) cost from the human perspective while, the 2 other constraints relate to the cost of using and maintaining the proposed solution from the perspective of the infrastructure used.

It is worth noting that this set of constraints is fairly challenging to meet as it prevents us from using language specific feature extraction methods or make use of special preprocessing approaches specifically designed for a given industry. For instance, we could imagine matching messages containing the word SIM-card together when dealing with messages from the telecommunications industry. Unfortunately, this would probably not work with data from other industries.

4.4 Clustering

From our previous description of the problem we are trying to solve and the definitions we introduced in the previous chapter, it becomes clear that we need to make use of an unsupervised machine learning approach in order to be able to find the main topics of our chat data without any use of human annotations. In other words, we would like to group our messages based on their content so that customer messages relating to the same topic would belong to the same group. To achieve this goal, we will make use of clustering techniques.

The goal of clustering is to group data by perceived or measured characteristics of similarity without making use of annotations or labels [47]. It has been applied to different domains such as gene expression patterns [10] or in the detection of network intrusions [53].

When grouping data points in a way that each point belongs to only one cluster, we perform what is known as hard clustering. On the other hand, if a given data point is allowed to be part of multiple clusters, we perform what is called soft clustering. For instance, one piece of text could contain different topics thus, it should be categorized into more than one category of topics [61].

Based on the previous definition, we can clearly see that clustering matches perfectly with the objective we are trying to achieve. Moreover, we will be looking at hard clustering approaches as our current supervised systems work in a way that one message has one annotation.

Like any other machine learning approach, applying clustering requires the identification of different metrics and features to use i.e how do we know how well different techniques perform with respect to each other and how to feed our data to a clustering algorithm. We will, in what follows, cover the most important aspects to consider when it comes to clustering texts in general and customer service messages in particular.

4.4.1 Feature Extraction

As mentioned earlier, depending on the task at hand, different feature extraction techniques can be used when dealing with textual data. Even though several techniques exist, one needs to keep in mind the trade-off between the representation power of the feature extraction techniques and their computational efficiency [71].

Bag-of-Words

One of the most common ways to represent textual data for clustering is the bag-of-words technique [71]. This family of approaches keeps track of the words appearing in a corpus of texts (set of customer messages) and represents each document (a single customer message in our case) as a vector of dimension $|V|$ where V is the set of words that appear across all documents in our collection (a customer's historical chat data). Each entry in the vector keeps track of a value related to a given word in our vocabulary like the word (term) frequency $TF(w, d)$: the frequency of word w in document d.

The main advantage of the bag-of-words approach is that it is simple from the conceptual perspective however, it fails to capture several grammatical and word-order information hence, the term "bag".

For instance, if we had a collection with two messages as follows:

- Message1 = "I need help to reset my password"
- Message2 = "Help me reset my password"

Based on the previous messages, our vocabulary V would become:

$$V = \{"I", "need", "help", "to", "reset", "my", "password", "Help", "me"\}$$

We would like to stress the important role of pre-processing our messages. Indeed, the words "help" and "Help" were considered as two different words which they are not. In this example, we will not apply any pre-processing since our aim is to demonstrate the use of bag-of-words approaches.

Based on the previously obtained vocabulary V, we would represent our messages as follows:

- { "I"=1, "need"=1, "help"=1, "to"=1, "reset"=1, "my"=1, "password"=1, "Help"=0, "me"=0}
- { "I"=0, "need"=0, "help"=0, "to"=0, "reset"=1, "my"=1, "password"=1, "Help"=1, "me"=1}

From the previous example, we can clearly see that obtaining a bag-of-words representation is very simple yet, this representation might produce large and sparse vectors i.e large vectors where most of the entries are 0.

Now that we have obtained our representation, we can apply different operations to it. The simplest one would be to apply a $L2 - normalization$ where the $L2 - norm$ of a real-valued vector $\mathbf{x} = (x_1, x_2, ..., x_n)$ is:

$$|\mathbf{x}| = \sqrt{\sum_{i=1}^{n} x_i^2}$$

Using the formula above, our bag-of-words would become:

- { "I"=$1/\sqrt{7}$, "need"=$1/\sqrt{7}$, "help"=$1/\sqrt{7}$, "to"=$1/\sqrt{7}$, "reset"=$1/\sqrt{7}$, "my"=$1/\sqrt{7}$, "password"=$1/\sqrt{7}$, "Help"=0, "me"=0}
- { "I"=0, "need"=0, "help"=0, "to"=0, "reset"=$1/\sqrt{5}$, "my"=$1/\sqrt{5}$, "password"=$1/\sqrt{5}$, "Help"=$1/\sqrt{5}$, "me"=$1/\sqrt{5}$}

Another popular normalization is the TF-IDF (Term Frequency - Inverse Document Frequency) which gives more information about a word's relevance compared to the word counts [71]. The TF-IDF value for a word w in a document d can be obtained as follows:

$$TF - IDF(w, d) = TF(w, d) * IDF(w)$$

Where the $TF(w, d)$ is just the number of occurrences of the word w in document d. Given that $DF(w)$ is the number of documents where the term w appears, IDF is defined as:

$$IDF(w) = 1 + log_{10}(|V|/DF(w))$$

The intuition here is that a word appearing in few documents will have more discriminative power for the document at hand hence, it will have a higher IDF. Conversely, if a word appears in most documents, it probably is less descriptive of any given document and will consequently have a smaller IDF value.

If we apply the IDF scaling to our previous example, we will obtain:

- { "I"=1.954, "need"=1.954, "help"=1.954, "to"=1.954, "reset"=1.653, "my"=1.653, "password"=1.653, "Help"=0, "me"=0}
- { "I"=0, "need"=0, "help"=0, "to"=0, "reset"=1.653, "my"=1.653, "password"=1.653, "Help"=1.954, "me"=1.954 }

From the representation above, we can clearly see that the words appearing in both messages have a smaller weight than those appearing in only one message. While the importance of such a weighting system might not be visible through the example we gave, the power of this approach becomes more visible in larger data-sets.

Word Embeddings

We mentioned in chapter 2 the fact that language was symbolic and that there was no simple mathematical operation that allows to transition from one word to another related word (Ex: "King" to "Queen"). Given that many of the NLP tasks deal with words, many researchers tried to develop a word representation that tries to solve this problem and that reflects similarities and dissimilarities between words instead of considering them as a unique symbols [55]. Most research efforts trying to reach the aforementioned goal are based on Harris's distributional hypothesis that states that words which appear in similar contexts have a similar meaning [40]. Of course, the notion of similarity is multi-faceted thus, what is considered similar or not might depend on the data that is used for training the embeddings. For instance, if we are learning our vectors using data about animal species then the words "tiger" and "cat" would be closer to each other than "cat" and "dog. However, if we are dealing with more generic texts then, "cat" and "dog" might be more similar as they are both types of pets [36].

One of the most iconic tools in this family of techniques is the *Word2vec* software whose approach is described in [66, 67]. This approach achieved state-of-the-art results in several linguistic tasks [55]. The *Word2vec* tool contains 2 algorithms to produce word vectors: Skip-Gram (SG) and the Continuous Bag Of Words (CBOW). In this document, we will describe the Skip-Gram approach.

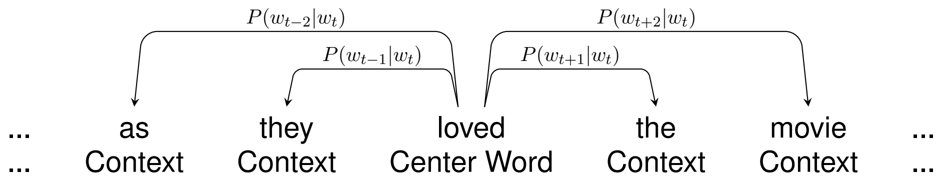
$P(w_{t-2}|w_t)$
$P(w_{t-1}|w_t)$
$P(w_{t+1}|w_t)$
$P(w_{t+2}|w_t)$
... as they loved the movie ...
... Context Context Center Word Context Context ...

Figure 4.1. *Description of the Skip-Gram approach*

Figure 4.1 contains an illustration of the Skip-Gram approach. The idea is to train a neural network that skims through the words of a training corpus and tries to predict their context (neighbouring) words up to a window size. For example, in Figure 4.1, we are trying to predict the context of the word "loved" with a window of size 2. The size of the window is a hyper-parameter of the algorithm. Essentially, the system will try to produce a set of parameters (our vector representation) such that it maximizes $P(ContextWords|CenterWord)$ for all the words in the training corpus.

More formally, given a training data-set D containing words w and their contexts c extracted from a text T (usually big), we would like to find parameters Θ for $P($Context Words$|$Center Wor
in order to maximize the corpus probability:

$$\underset{\Theta}{\operatorname{argmax}} \prod_{(w,c)\in D} p(c|w;\Theta)$$

with $p(c|w;\Theta)$ modeled as follows:

$$p(c|w;\Theta) = \frac{\exp(v_c \cdot v_w)}{\sum_{c'\in C} \exp(v_{c'} \cdot v_w)}$$

with v_c and $v_w \in R^d$ being the vector representations of the c and w while C is the set of all contexts. It is worth noting that one given word has 2 representations. For instance, the word "love" will have a word representation and a context representation hence the use of C here [37].

By introducing the log to the first equation, we obtain:

$$\underset{\Theta}{\text{argmax}} \sum_{(w,c)\in D} \log(p(c|w;\Theta)) = \sum_{(w,c)\in D} \left(\log(\exp(v_c \cdot v_w)) - log(\sum_{c'} \exp(v_{c'} \cdot v_w)) \right)$$

The main assumption in this work lies on the fact that a maximization of the previous equation produces quality word-vectors i.e words appearing in the same context will have similar vectors. As mentioned earlier, this approach has confirmed the assumption in practice as it achieved state-of-the-art-results on several tasks.

Because word2vec is usually trained on large data-sets such as Wikipedia, it becomes important to have an efficient approach for learning our word vectors, as a result, [67] introduced certain modifications to their learning objective in order to make it more efficient.

Given the success of the Word2vec approach, other techniques for word representations appeared such as *Glove* [77] which uses an explicit matrix factorization approach ([55] showed that even *Word2vec* is a type of matrix factorization approach) and *fastText* [12] which uses a similar approach to word2vec but introduces the idea of treating words as character n-grams. For example, the word "orange" would be obtained by summing the vectors for "<or", "ora", "ran", "ang", "nge", "ge>" where $<$ and $>$ are just start and end tokens and assuming that we decide to train using 3-grams.

The main advantages of the *fastText* approach compared to word2vec is that it generates better embeddings for rare words as we are more likely to see training data for n-grams compared to complete words. In addition, this approach allows to produce embeddings for words that were not in the training data.

Sentence and Short-Text Embeddings

We just covered concepts related to word embedddings. However, many classes of NLP problems deal with longer chunks of texts such as sentences or paragraphs. For instance, our problematic deals with customer service chat messages which typically are short sentences or paragraphs.

The simplest way to produce embeddings for this type of textual data is to use a naive word-embedding averaging i.e identify words in the text of interest, produce their respective embeddings using, for instance, word2vec then, obtain the text embedding by applying an element-wise averaging operation. This technique was, surprisingly, shown to work better than some more complex models such as neural networks based on Long

Short-Term Memory units [34] (LSTM) [75]. Based on this, some approaches attempted to find better weighting schemes for the word vectors such as in [4].

Techniques that go beyond the simple averaging have been developed like in *Skip-thoughts* [50]. This unsupervised approach can be thought of an extension of the *Word2vec* learning approach where, instead of trying to predict context words of a central word, we try to predict context sentences of a central sentence. This approach uses Recurrent Neural Networks and, while interesting, is very slow to use and requires a lot of resources to train, based on our experiments. In addition, it seems that it is not well suited to sentence similarity tasks [75] which might be an issue in our case as we are trying to group similar messages together.

Another unsupervised sentence embedding approach is called *Sent2vec* [75] which is an extension of the Word2vec approach. Indeed, this approach is based on the Continuous Bag Of Words (CBOW) algorithm of Word2vec where contrary to the Skip-Gram approach we try to predict a center word from its context instead of the context of a center word. This means that during the training phase, the vectors for the context words are averaged and the the dot product between the context average and the target word vector is minimized. The authors of the *sent2vec* model argue that such an approach produces vectors that are optimized for a sequence of words. Based on this, the authors propose to modify the approach and use an intuitive approach and train on linguistically meaningful context windows such as sentence or paragraph boundaries instead of using fixed window sizes. An additional contribution is to use a *fastText*-like approach where word n-grams are used instead of character n-grams thus, improving the model's ability to produce quality vectors for different combinations of words.

In addition to the unsupervised approaches for creating sentence and short-text embeddings, supervised approaches have been devised such as in [16, 23]. The idea of these approaches is to train embeddings using annotated datasets for tasks like Natural Language Inference (NLI) in order to improve the embedding quality.

Infersent [23] is one type of supervised sentence embedding technique. In this work, the authors used Standford's Natural Natural Language Inference corpus [13]. The corpus contains around 570,000 human-written and labelled English sentence pairs. The sentence pairs can have one of 3 labels: entailment, contradiction and neutral.

Table 4.1 contains examples extracted from the SNLI where the judgments are produced by 5 Mechanical Turk workers and the overall judgment is selected as the consensus judgment.

Based on the SNLI data-set, the developers feed the Text and hypothesis to a neural network architecture. This architecture first encodes each of the Text and hypothesis using a BiLSTM with max pooling encoder thus, producing vectors u and v for the text and hypothesis respectively. After this, it feeds the information coming from the two sentences as a vector containing: a concatenation of u and v (u, v), an element-wise product $u * v$

Text	Judgments	Hypothesis
A man inspects the uniform of a figure in some East Asian country.	contradiction C C C C C	The man is sleeping
An older and younger man smiling.	neutral N N E N N	Two men are smiling and laughing at the cats playing on the floor.
A soccer game with multiple males playing.	entailment E E E E E	Some men are playing a sport.

Table 4.1. *Examples from the Stanford Natural Language Inference (SNLI) corpus.*

and an absolute element-wise difference $|u - v|$ to classifier with fully-connected layers ending with a softmax function.

The *infersent* produced embeddings allowing to obtain state-of-the-art results (beating techniques like skip-thoughts) on several linguistic tasks. This shows the potential of supervised approaches to learning embeddings for sentences and short texts. However, this strength might also be a liability as corpora as large as the SNLI might not be available for several languages.

4.4.2 Clustering Algorithms

Once we have chosen our feature extraction approach, we can feed our data to a clustering algorithm. There are different types of clustering algorithms, making certain assumptions about our data and requiring the selection of different parameters. We will describe here three of them: K-Means [60], DBSCAN [30] and HDBSCAN [15].

K-Means

K-Means is a one of the earliest clustering approaches. It is based on the idea of partitioning the data into a number of exclusive partitions or clusters [39]. The idea of this family of techniques is to partition n data points into K partitions (with the condition that $K \leq n$) using different criteria.

More specifically, provided that we have a data-set D containing n points, a partitioning method would place the points in D into K clusters: $C_1, C_2, ..., C_K$ which means that $C_i \subset D$ and that $C_i \cap C_j = \emptyset$ for $(1 \leq i, j \leq K)$.

The K-Means approaches the partitioning of the data by considering each cluster C_i as being represented by its *centroid*, in other words, by its center point which, in the case of K-Means, is defined as the mean of the data points belonging to the cluster:

$$m_i = \frac{1}{|C_i|} \sum_{x \in C_i} x$$

Based on this, K-Means will try to minimize the within-cluster sum of squares E defined as follows:

$$E = \sum_{i=1}^{K} \sum_{x \in C_i} dist(x, m_i)^2$$

Where $dist(x, m_i)$ represents the Euclidean distance between a point and its cluster centroid.

More formally, K-Means can be described by the following procedure:

1. Arbitrarily select K points as the initial cluster centroids,
2. **Repeat:**
3. (Re)assign each point to the closest cluster centroid (from the Euclidean distance perspective),
4. Update the cluster centroids based on the points belonging to the cluster,
5. **Until no change in the cluster assignments**

One of the main advantages of the K-Means algorithm is its simplicity and ease of implementation. However, it requires to choose the number of partitions K which we typically do not know. In addition, it is sensitive to the choice of the initial cluster centroids i.e two different runs of K-Means might produce significantly different results. Moreover, the algorithm might suffer from outliers (points that are distant from the rest of the data) because they might drastically distort the value obtained for the cluster centroid.

DBSCAN

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [30] is a density-based clustering approach, in other words, it considers clusters as regions of high data density separated by regions of low data density hence, considering points not belonging to a high-density region as being noise points thus, not assigning them to any cluster. Contrary to the K-Means approach, DBSCAN can produce clusters of any shape and does not necessarily favor more convex clusters that revolve around a centroid.

Figure 4.2 contains an illustration of the previous point where running K-Means with $K = 2$ produces the results visible on the left sub-plot while results obtained with HDBSCAN can be seen on the right sub-plot. We can clearly see that K-Means tries to produce clusters whose points revolve around a center while DBSCAN produces clusters that naturally follow the density of the data.

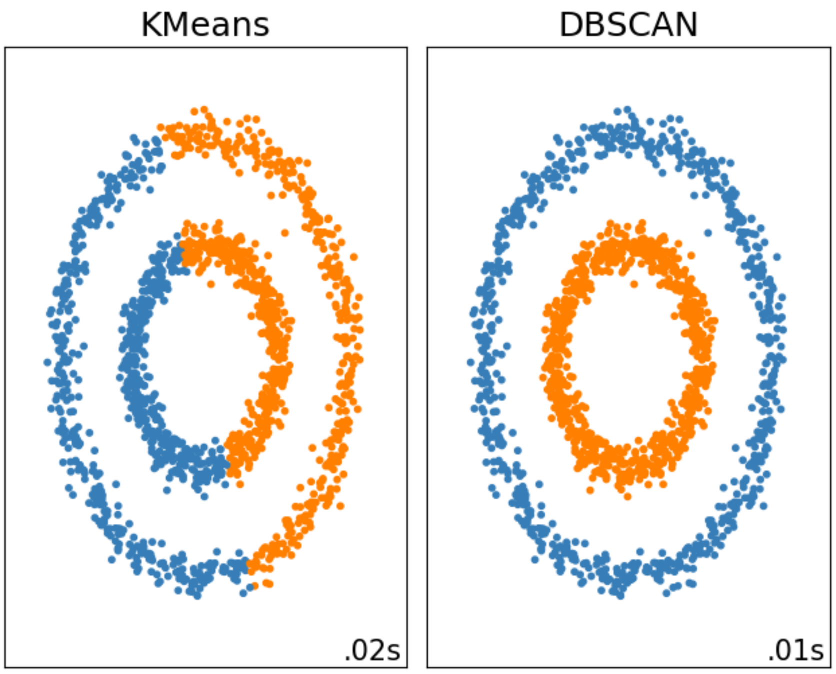

Figure 4.2. *Clustering results obtained with KMEANS and DBSCAN for circular data*

In order to find regions of density, DBSCAN uses the notion of *core point* where a core a point is any point having *minSamples* (other points) within a radius of ϵ with *minSamples* and ϵ being two user-defined parameters. This means that a core point and its neighbours are belonging to a dense area.

Once core points have been identified, a cluster is formed by connecting a core point to its neighbouring core points which are, in-turn, connected to their respective neighbouring core points until no core point is found in the neighbourhood. Neighbours of a core point that are not core points also belong to the same cluster as the core point. The points that are neither core points nor in the neighbourhood of a core point are considered as noise points.

One of the main advantages of DBSCAN is that it has the ability to produce clusters with arbitrary shapes. However, DBSCAN, is not so good at clustering data with different densities as the choice of *minSamples* and ϵ is done for the whole data-set and not for specific regions. In addition the results obtained with DBSCAN are highly sensitive to the choice of parameters which makes it hard to work with even if it has the potential of achieving satisfactory results in theory.

HDBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [15] is an extension of DBSCAN that turns it into a hierarchical clustering algorithm. One of the main motivations is to deal with DBSCAN's inability to handle data having varying density which can potentially be useful in our the context of our problematic.

HDBSCAN keeps the *minSamples* parameter from DBSCAN. Using this parameter, the notion of core distance $core_{minSmaples}(x)$ for each point in the data-set is calculated. The core distance is just the distance between a point x and its neighbour at position *minSamples*. In practice, it means that if *minSamples = 5* then, the core distance of a point x would be the distance between x and its 5th nearest neighbour as we can see in Figure 4.3 [64].

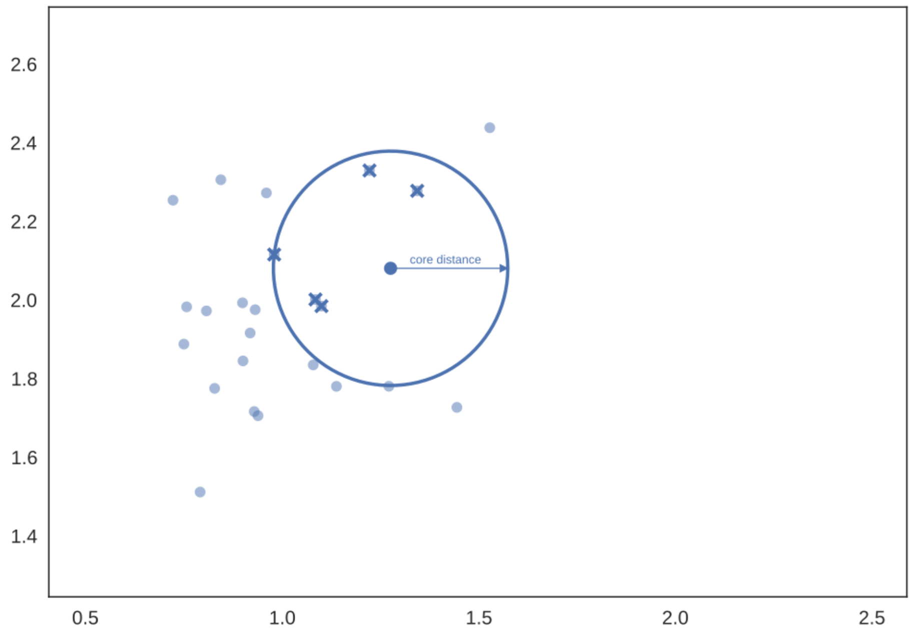

Figure 4.3. *Finding the core distance of a point*

Once the core distance has been computed for each point in the data-set, the pairwise distance between two points x and y is redefined in order to make points in low density areas farther away and keep points in high-density regions at the same distance. This redefined distance is called mutual reachability distance and is formulated as:

$$d_{mreach}(x, y) = max\left\{core_{minSmaples}(x), core_{minSmaples}(y), d(x, y)\right\}$$

Where $d(x, y)$ is the original distance between x and y using, for instance, the Euclidean distance. The next step of the algorithm consits in considering a graph consisting of the data points as vertices connected by edges whose weight is the mutual reachability distance between the two points. Once the tree is built, a Minimum Spanning Tree (MST) is extracted from it in order to obtain the smallest weight graph that connects all the vertices together without cycles.

When we obtain the Minimum Spanning Tree, we can create a hierarchy of points by ordering the edges of the MST in increasing order of weight then, connecting pairs of points while iterating through the weights until we connect all the points. Figure 4.4 [64] contains an example of the obtained dendrogram (diagram representing a tree) form the aforementioned process.

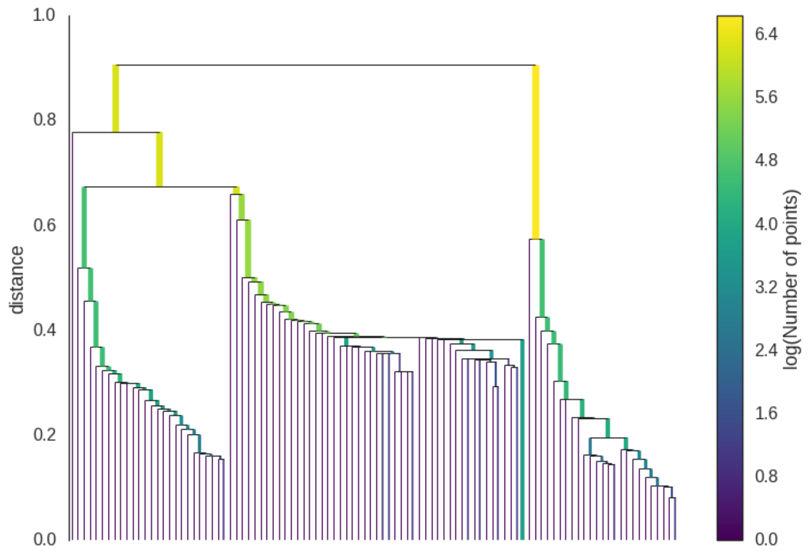

Figure 4.4. *Hierarchy obtained by connecting data-points based on their minimum reachability distance*

Now, we can use the second user-specified parameter of the algorithm i.e *minClusterSize* which tells the size of the smallest cluster the user expects. The cluster hierarchies were created in a bottom-up way (from leaves to root), now, we will read the dendrogram in a top-down manner (from root to leaves) while highliting the moment a cluster was created (where a data split happened) and when it stopped existing (where its number of points fell below the *minClusterSize*. An example result of this process can be seen in figure 4.5 [64] which contains a condensed cluster tree.

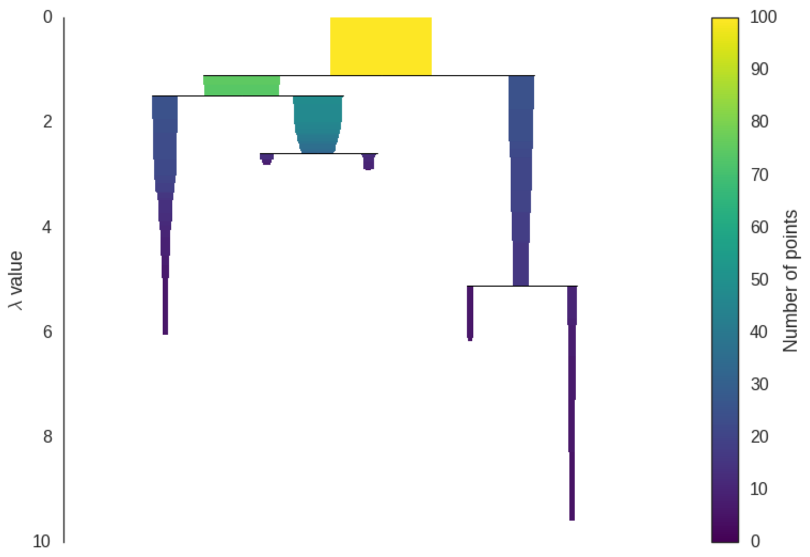

Figure 4.5. *Condensed hierarchy tree*

The last step of the algorithm consists in extracting the actual clusters from the candidate clusters in the condensed cluster tree that we can see in Figure 4.5. Choosing a cluster means that we cannot choose any of its descendants. Essentially this step is about finding clusters with the biggest life span (or with the largest area of ink in the condensed cluster tree in Figure 4.5) thus, obtaining the result visible in Figure 4.6 [64]. Points that do not belong to any cluster are considered as noise points noise.

Formally the last step consits in using $\lambda = \frac{1}{distance}$ where the distance is just the mutual reachabiliy distance. Based on this we can define λ_{birth} and λ_{death} as the λ values when the cluster started to exist (became a cluster) and ceased to exist (created clusters that are smaller than *minClusterSize*) respectively. We can also define a $\lambda_{birth} < \lambda_p < \lambda_{death}$ for each point in the cluster and which refers to the λ at which the point fell out of the cluster.

Now, we can define a cluster stability value for each cluster as:

$$\sum_{p \in Cluster} \lambda_p - \lambda_{birth}$$

Using the previous formula, we start by considering all the leaves of the hierarchy tree as being selected clusters then, we go up the tree and check whether the sum of stabilities of the child clusters is higher than that of the cluster being processed, then, the cluster stability is set as being the sum of its children stabilities. On the other hand, if the cluster stability is bigger than the sum of cluster stabilities of its children then, the cluster is selected and all its descendants are un-selected.

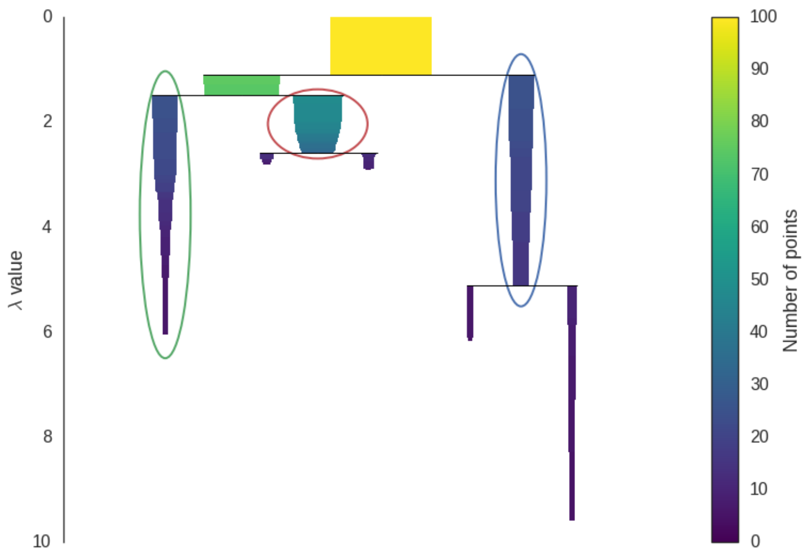

Figure 4.6. *Extracting clusters from a condensed hierarchy tree*

It can be see that HDBSCAN, similarly to DBSCAN, produces clusters of arbitrary shapes however, unlike DBSCAN, it allows to find clusters of different densities for one data-set. In addition, it reduces the burden of finding the ϵ value which feels less intuitive to choose than the *minClusterSize*.

4.4.3 What Makes Messages Close: Distance Metrics

We have been discussing the fact that clustering aims at grouping data points together based on a perceived or measured similarity criteria. In addition, we clearly saw from our description of some clustering algorithms that distance measurements are an integral part of their work. Based on this, one might wonder, what distance metric to use in order to assess how close 2 data points are (2 customer messages). We will discuss here some of the most popular approaches to measuring distance between two points.

Euclidean Distance

Euclidean distance is one of the simplest and more natural ways of computing the distance between two points. Essentially, it measures how long is the straight line between these points. This distance metric has been used with clustering algorithms such as K-Means [39].

The Euclidean distance between points $x = (x_1, x_2, ..., x_d)$ and $y = (y_1, y_2, ..., y_d)$ is measured as follows:

$$d_{Euclidean}(x, y) = d_{Euclidean}(y, x) = \sqrt{\sum_{i=1}^{d}(y_i - x_i)^2}$$

The Euclidean distance is quite intuitive and simple to implement. In addition, it can be computed relatively quickly. However, it can suffer if the different dimensions of our vector have different units. For instance if we had 2 points with 3 dimensions: age, weight and salary, it is quite obvious that the salary dimension might look more important than it really is because salary values are in the range of hundreds or thousands while the age and weight of a person rarely exceed 100. In addition the Euclidean distance can potentially fail when applied to high-dimensional data [27].

Cosine Distance

Cosine distance is one of the most common and effective techniques when it comes to measuring the distance between words or short-texts in vector space [31, 36]. Contrary, to the Euclidean distance, it does not give importance to the magnitude of the different dimensions in the space.

The cosine distance for vectors x and y can be calculated as follows:

$$1 - cossim(x, y) = 1 - \frac{x \cdot y}{||x|| \cdot ||y||}$$

With $cossim(x, y)$ being referred to as *cosine similarity*.

It worth noting that the Euclidean distance and the cosine distance are equivalent when data vectors are normalized to have a unit norm [92]. More specifically, the ranking produced by the Euclidean distance and the cosine distance will become the same. Indeed the Euclidean distance $||x - y||$ is related to the cosine distance as follows:

$$||x - y||^2 = (x - y)^T(x - y) = ||x||^2 + ||y||^2 - 2x^T y$$

Given that the vectors are normalized i.e $||x||^2 = ||y||^2 = 1$, we obtain:

$$||x - y||^2 = 2(1 - cos(x, y)) = 2 * cosineDistance(x, y)$$

In other words:

$$EuclideanDistance(x, y) = \sqrt{2 * cosineDistance(x, y)}$$

Metric Learning

The distance metrics we just described are quite generic as they can be and have been used on wide variety of data. In addition, it has been shown that learning a good distance metric can improve the performance of several machine learning tasks including clustering tasks [92]. This leads to the question: Is there a way to have a metric that is specific to our data and use case?

Because manually designing metrics based on a specific use case is tedious at best, techniques that attempt to automatically learn a metric from data have been devised and generally fall under the denomination of metric learning techniques [9].

Given a data-set D consisting in points having the same dimension d. If we take points $x, y, z \in D$ then, $M : D \times D \to R$ is called a distance metric if it satisfies the following properties [92]:

- Nonnegativity: $M(x, y) \geq 0$
- Coincidence: $M(x, y) = 0 \iff x = y$
- Symmetry: $M(x, y) = M(y, x)$
- Subadditivity (triangle inequality): $M(x, y) + M(y, z) \geq M(x, z)$

It is worth noting that, if the coincidence constraint is relaxed, then M is called a pseudo-metric. In general, metric learning techniques try to adapt a pairwise real-valued metric function like the Mahalanobis-like distance [1] [93]:

[1]The actual Mahalanobis distance uses Σ^{-1} the inverse of the covariance matrix between x and y: $(\sqrt{(x - y)^T \Sigma^{-1}(x - y)})$

$$d(x,y) = d_A(x,y) = ||x-y||_A = \sqrt{(x-y)^T A(x-y)}$$

It is easy to see that if $A = I$ the identity matrix then, the distance will be equivalent to the Euclidean distance. In addition, in order for $d_A(x,y)$ to be a metric (a pseudo-metric) then, A needs to be a Positive Semi-Definite matrix. Moreover, an interesting feature about this family of metrics is that for two points x and y, we have:

$$EuclideanDistance(A^{1/2}x, A^{1/2}y) = \sqrt{(x-y)^T A(x-y)}$$

In other words, this type of metric is equivalent to applying a Euclidean distance metric on a rescaling of the our data pints [93].

There are different ways to learn the matrix A. The approaches usually use supervision to achieve this goal. They can achieve this by using actual labels from an annotated data-set or just use side-information such as pairwise constraints that tell whether a pair of points belong to the same class or not [9]. After this, they learn a matrix A with the constraint that it is Positive Semi-Definite.

One of the earliest approaches in this family techniques is the Mahalanobis Metric learning for Clustering (MMC) [93] which tries to optimize a convex objective function with no regularization. Specifically, given two sets of constraints S and D:

$$S = \{(x_i, x_j) : \text{where } x_i \text{ and } x_j \text{ belong to the same class}\}$$
$$D = \{(x_i, x_j) : \text{where } x_i \text{ and } x_j \text{ belong to different classes}\}$$

The MMC approach will try to maximize the sum of distances between points from different classes and minimize the sum of distances between points from the same class as follows:

$$\min_A \sum_{(x_i,x_j)\in S} ||x_i, x_j||_A^2$$
$$\text{s.t} \sum_{(x_i,x_j)\in D} ||x_i, x_j||_A \geq 1,$$
$$\text{A is PSD}$$

The authors mention that the choice of the constant 1 is not important. The algorithm used to solve the problem is a projected gradient approach whose main challenge is the need to perform an eigenvalue decomposition of A for each iteration hence, making the approach not easily scalable.

4.4.4 Clustering Evaluation

Given the different options of feature extraction, clustering algorithms and similarity metrics, we will need an objective criteria to evaluate which approach works the best for our data. Although qualitative evaluation using human experts is possible, it is in practice difficult to put into place and not reproducible as humans might produce an opinion based on their knowledge of the field or personal preference at a given time.

While quantitative performance evaluation of supervised approaches is straight forward as we have ground truth labels attached with our data, there is no agreed upon standard for evaluating clustering experiments thus, making evaluation efforts difficult [46].

Nevertheless, several methods for evaluating the adequacy of a produced clustering structure having devised. These methods fall into 3 categories [48]:

- External performance criteria: that evaluate the given clustering results against ground truth information i.e how well clustered items correspond to real annotations,
- Internal performance criteria: that evaluate the quality of the clustering results by only making use of the cluster results themselves i.e no external data is used. Typically, these metrics compute such things as cluster compactness meaning how dense points in a given cluster are, and cluster separation meaning how well different clusters are separated from each other,
- Relative performance criteria (which could be considered as a special case of internal criteria): that compare different cluster structures and tells which one would be better. For instance, this can be useful to evaluate different sets of hyperparameters,

From the previous definitions, we can clearly see that external metrics, even if they are potentially the most reliable, are difficult to apply in a real-world setting. Indeed, the idea of unsupervised learning in general and of clustering in particular is to be able to explore the data without needing to have ground truth labels. Furthermore, a given annotation could just represent one way to annotate the data i.e not be the only possible way to structure a data-set.

When it comes to internal and relative performance criteria, they are more reasonable from the conceptual perspective as they do not require any sort of annotation to evaluate a cluster structure however, these metrics can easily be tricked by certain types of cluster structures. Indeed, we could artificially boost the obtained metrics by intentionally applying some pre-processing tricks or enforcing certain constraints making our clusters more compact or having better separation thus looking better from a metric perspective. This means that good scores do not necessarily mean good clustering.

Moreover, it is worth mentioning that some clustering evaluation criteria tend to favor certain cluster structures regardless of their actual quality. Indeed, we mentioned earlier that K-Means produces clusters using the notion of *centroid* thus, producing convex-shaped clusters while an algorithm like DBSCAN can produce arbitrarily-shaped clusters. Similarly, some metrics might give a higher score to, say, convex clusters just because they are convex and not because they are actually better.

From the aforementioned descriptions, we can see that clustering evaluation is hard as we are essentially trying to tell how good a clustering output is without necessarily knowing what good means in our context.

Now that we have covered clustering evaluation criteria from a high-level perspective, we will introduce some of these criteria in more detail in what follows.

Adjusted Rand Index

The Adjusted rand Idex (ARI) [45] is an external clustering evaluation metric based on counting pairs. The index measures the similarity between the ground truth labels and the assignment obtained by a clustering approach. The ARI produces values in $[-1; 1]$ where 1 represents a complete match with the ground truth assignment and 0 represents random (uniform) assignments. One of the main advantages of the Adjusted Rand Index is that it does not make any assumption regarding cluster structures.

Before giving the mathematical formulation of the ARI, we will start by giving the formulation of the Rand Index (RI). Given L the ground truth label assignment and K the clustering assignment for our data points, the RI is given as follows:

$$RI = \frac{a + b}{C_2^n}$$

Where a is the number of pairs of elements (data points) that are in the same cluster in L and in K while b is the number of pairs of elements that are in different clusters in L and K. Finally, C_2^n represents all the number of possible pairs of points in the dataset.

Some of the issues with previous formulation is the fact that it gives values close to 1 (meaning good clustering) when the number of clusters increase and it does not guarantee that we get a value close to 0 for random clustering results [83]. To tackle such issues, the Rand Index is modified to account for chance thus, obtaining the Adjusted Rand Index:

$$ARI = \frac{RI - E[RI]}{max(RI) - E[RI]}$$

Adjusted Mutual Information

Another external evaluation metric for clustering is the Adjusted Mutual Information (AMI) [91]. Similarly to the ARI which is a modified Rand Index to account for chance i.e , the AMI is based on the notion of Mutual Information (MI) modified to account for chance where the notion of Mutual Information measures the mutual dependence between a pair of random variables. The AMI has its values in the $[0, 1]$ interval and, similarly to the ARI, gives values close to 0 for random label assignments.

Formally the Mutual Information between two partitions of our data $U = \{U_1, U_2, ..., U_T\}$ and $V = \{V_1, V_2, ..., V_C\}$, where U and V are two different clustering partitions having T clusters and C clusters respectively (in the context of clustering evaluation, one of the partition represents the ground truth (real) partitioning of the data and the other contains the clustering results), is given as follows:

$$MI(U,V) = \sum_{i=1}^{T} \sum_{j=1}^{C} p(i,j) log(\frac{p(i,j)}{p(i)p'(j)})$$

With $p(i) = \frac{|U_i|}{N}$ representing the probability that a randomly picked data point belongs to cluster i in U, $p'(j) = \frac{|V_j|}{N}$ representing the probability that a randomly picked point belongs to cluster j in V and $p(i,j) = \frac{|U_i \cap V_j|}{N}$ representing the probability that any given point in our data belongs to clusters i and j in U and V respectively.

Building on the previous definition, we obtain a definition for the Adjusted Mutual Information (similar to that of the Adjusted Randi Index) as follows:

$$AMI(U,V) = \frac{MI(U,V) - E[MI(U,V)]}{max(H(U), H(V)) - E[MI(U,V)]}$$

Such that:

$$H(U) = -\sum_{i=1}^{T} p(i) log(p(i)) \text{ and } H(V) = -\sum_{i=j}^{C} p'(j) log(p'(j))$$

Where $H(U)$ and $H(V)$ represent the entropy associated with partitions U and V respectively.

Silhouette Coefficient

Silhouette Coefficient or Silhouette Index [82] is an internal clustering evaluation metric i.e it does not require ground truth clusters which is useful in most clustering cases. It can be used for example to identify the most appropriate number of clusters when using algorithms that need an a priori number of clusters to be set such as K-Means.

The silhouette coefficient is defined as follows:

$$SC = \frac{b - a}{max(a, b)}$$

Where a is the mean distance between a point and other points in the same cluster and b the mean distance between a point and other points in the following nearest cluster.

One of the advantages of the Silhouette Coefficient is that it tends to give higher scores to dense and well separated clusters however, it tends to give a higher score to convex clusters and might, consequently, unjustly penalize results produced by a density-based algorithm.

BCubed

BCubed precision and Bcubed recall are external clustering evaluation metrics. They were originally defined as algorithms in [5] and have been described as a function in [3]. The idea of these metrics is to estimate precision (and recall) for each item in our sets unlike other external evaluation metrics which evaluate scores on the set as whole. Before being able to introduce the BCubed precision and recall, we need to introduce the concept of correctness:

$$Correctness(i, i') = \begin{cases} 1, & L(i) = L(i') \iff C(i) = C(i') \\ 0, & \text{otherwise} \end{cases}$$

Where $L(i)$ and $C(i)$ tell the label (ground truth) and Cluster of an item i. In essence, the correctness tells that two items are correctly related when they have the same label if and only if they are in the same cluster. Building on top of this, the BCubed precision and BCubed recall can be defined as:

$$BCubedPrecision = Avg_i[\,Avg_{i'.C(i)=C(i')}[\,Correctness(i, i')]\,]$$

$$BCubedRecall = Avg_i[\,Avg_{i'.L(i)=L(i')}[\,Correctness(i, i')]\,]$$

It is easy to combine the BCubed Precision and BCubed recall using the F1-measure:

$$\text{F1-Bcubed} = 2 * \frac{BCubedPrecision * BCubedRecall}{BCubedPrecision + BCubedRecall}$$

Even if the computation of the F1-BCubed can be computationally heavy, it should not be a problem as it is an external evaluation metric which means that it will be ran on reasonably-sized data-sets. One of the main advantages of the F1Bcubed metric is the fact that, contrary to other metrics such as the ones based on Mutual Information and the

Rand Index (which satisfy only part of the constraints), it satisfies 4 constraints regarding clustering results [3]:

1. It gives higher scores to more homogenous clusters i.e clusters whose points belong to mostly one cluster,
2. It gives higher scores to clustering results that group as many points from one class as possible in the same cluster,
3. Penalizes less clustering results that introduce disorder to an already disordered cluster,
4. Prefers small errors in big clusters to many smaller errors in smaller clusters,

4.5 Related Works

Clustering has been widely applied and studied in the text domain. It can have several applications such as data visualization, document organization or topic detection [2]. While many of the current research efforts towards clustering textual data deal with relatively long and clean texts like news articles such as in [52, 86] or with short-text data from SMS messages or Twitter posts such as in [70, 79], as far as we are aware of, no approach deals specifically with customer service chat data which, contrary to news articles, is typically shorter and contains a lot of spelling mistakes and unlike SMS, Twitter or similar social media texts is not limited in size. Indeed, customer service messages are usually short texts but have no size limit and, on some occasions can span several sentences. Yet, we believe that short-message clustering is the closest effort towards dealing with customer chat data.

Given the increasing popularity of social media platforms such as Twitter, short-text clustering is attracting the interest of the research community. The main difference between short-text clustering and clustering longer chunks of text lies in the fact that words usually appear only once in a given short text (important words and less important words have the same frequency) making term-frequency approaches less discriminative of a given document thus, meaning a more difficult clustering task.

In order to deal with the aforementioned problem, some approaches make use of additional data in their clustering approach. For instance, [7] makes use of Wikipedia data in order to enrich their textual features. More specifically, Wikipedia articles are dumped and pre-processed. Then, for each short message to be processed, 2 feature extraction techniques are used. Based on the two representations, the researchers look for the best matching Wikipedia articles and use their titles as additional features for the short text at hand hence, virtually lengthening the short-text. While this approach seems interesting as it makes use of easily available data, it requires the manipulation and pre-processing of large amounts of data. In addition, the experiments were only carried out for English which probably has more articles than other langauges.

Another approach uses ontologies in order to improve clustering results [33]. This approach involves, among other things, the disambiguation of polysemic nouns i.e nouns with multiple meanings. Specifically, nouns a replaced by their most appropriate senses based on the context of the document. For instance, the word "cat" has several meanings in WordNet (a lexical database for English). If "cat" is used alongside words like "kitten" then, it is represented by the concept of "feline mammal". Conversly, if the word appears alongside words like "construction", it is more likely to refer to a "Caterpillar" (a vehicle used in construction works). Here too, the approach relies on the use of external data which, in this case, requires linguistic understanding and knowledge in addition to the manipulation of external data.

The approach described in [79] deals with English Twitter messages. The approach uses some assumptions that cannot be generalized to other types of data. Specifically, the authors consider that tweets containing the same URL refer to the same topic and that a tweet and a reply to it refer to the same topic. As mentioned earlier, our constraints do not allow us to use specific features in the data as our approach needs to be as generic as possible.

A more generic approach to short text clustering can be found in [94]. In this approach, a Convolutional Neural Network (CNN) is used to learn a deep feature representation of the short-text messages that is then used with a K-Means clustering algorithm. The CNN architecture is fed with a weighted average of the embeddings (word2vec was used) of the words in each short texts. The weights were obtained from the Term-Frequency values or TF-IDF values from the TF and TF-IDF representation of the texts. The target of the Convolutional Neural Netowrk is obtained by binarizing the results of the Locality Preserving Indexing algorithm applied on the BoW representation of the short texts.

The authors of the previous approach report an improvement in clustering results using their approach. While the described method is nice in the sense that it does not require any external data, it still represents a quite heavy overhead since it requires heavy operations for preparing the data, for the CNN training and also for the training itself. In addition, the approach was tested on 3 data-sets which do not really represent the content of customer service data.

Overall, many of the approaches dealing with text clustering handle data that is different from customer service chat data. In addition, many of these approaches require the use of external data which is not necessarily available for multiple languages (like WordNet) or requires linguistic knowledge. Furthermore, the use of external data involves a potentially large overhead as, for instance, the manipulation of Wikipedia dumps is demanding in terms of processing resources. Moreover, some approaches make use of tricks specific to a certain type of data like we have seen with the Twitter approach.

4.6 Conclusion

In this chapter, we introduced the context of this work and the problem we are trying to solve. In addition, we described the family of techniques that can be used to solve this problem and the moving pieces that go with them (evaluation, feature extraction, clustering algorithms and distance metrics). Finally we concluded this chapter by an overview of the approaches that try to deal with a similar problematic.

5 EXPERIMENTS AND RESULTS

5.1 Introduction

After having introduced and covered the basic building blocks of our problematic, we will, in this chapter, cover aspects regarding our approach. More specifically, we will describe the methodology we followed towards achieving our goal, the evaluation metric we used as well as the data for our experiments. Finally, we will expose our results, discuss some of our findings and propose a clustering set-up that significantly improves over our baseline.

5.2 Methodology

We mentioned in the previous chapter that there are two types of quantitative evaluation metrics for clustering (we consider relative evaluation as a part of the internal metrics family): metrics that require ground truth annotations (external metrics) and metrics that do not require any ground truth annotation (internal metrics).

Given no recommendation as to what type of metric to use, one would be tempted to choose an internal metric for cluster evaluation as it would allow us to assess the quality of our approach in a real-life setting, i.e where no annotations are available. However, as we have seen with the Silhouette Coefficient and we found with other internal metrics, there is, most of the time, a preference towards a certain shape of clusters (usually convex clusters produced by an algorithm like K-Means) thus, preventing us from comparing the results of different clustering algorithms. Some metrics have been specifically developed to deal with the output of density-based algorithms such as Density-Based Clustering Validation (DBCV) [69] however, our early experiments showed that this metric does not scale with the size of our un-annotated data-sets. In addition, such a metric would not necessarily mean that it would treat convex-shaped results equally. At the same time, an extensive qualitative assessment of our multiple clustering runs was out of the question as it would require the constant availability of the whole annotation team for the assessment to happen in time not to mention the high level of subjectivity of these evaluations.

With these limitations, and given the fact that we already possess several annotated data-sets, we decided to use an external evaluation metric for our clustering runs with some of our already annotated customer data. The objective of the experiments being to explore a large space of feature extraction techniques, clustering algorithms and similarity metrics and assess whether one of them improves over a baseline that was in use in the company.

Considering the variety of external evaluation metrics, we are still faced with the challenge of finding the one that corresponds to our definition good clustering. Our initial candidates were the Adjusted Rand Index (ARI), Adjusted Mutual Information (AMI) and F1-BCubed. The two first were chosen because they are widely used in the clustering literature [81]. The last one because it seemed to satisfy more cluster quality constraints than the former metrics [3].

Our initial ARI scores were hard to interpret as most of the values between different configurations (feature extraction, algorithms etc...) were very close to each other. From these initial results and the recommendation in [81] that specifies that the ARI should be used when we are dealing with large clusters (ground truth classes) having equal sizes whereas the AMI should be selected when the clusters (ground truth classes) are unbalanced and there are small clusters, we decided to drop the ARI. Indeed, our data is mostly unbalanced as, for instance, it is natural for greetings to outnumber other messages. Additionally, categories themselves might have different frequencies depending on the concerns of the clients.

As for the selection between the Adjusted Mutual Information and the F1-BCubed, we decided to settle the choice by means of qualitative evaluation. We ran a series of tests covering a large configuration of parameters for 2 of our data-sets in 2 different languages then, we gave the top result according to the AMI and the top result according to the F1-BCubed metric to a panel consisting of all the 3 members of ultimate.ai's annotation team meaning $3 * 2 = 6$ possible opinions.

	Top F1-Bcubed	Top AMI	Neutral
Cluster purity	5	1	0
Cluster completeness	6	0	0
Result processing time	6	0	0

Table 5.1. *Qualitative evaluation of the AMI selection compared to the F1-Bcubed selection.*

Table 5.1 contains results of our qualitative evaluation aiming to determine which evaluation metric is more suitable to evaluate our clustering runs using our data. The criteria used for the evaluation were chosen after a discussion with the annotation team and are partly based on criteria used in [3]. Cluster purity means how clean a cluster is, in other words, whether a majority of the messages in one given cluster belong to the same ground truth class. The Cluster completeness tells whether 1 ground truth class is to be mostly found in 1 cluster as splitting one class across multiple clusters is not desirable. The last criterion is about the processing time of the clustering result: the faster, the better.

The results in Table 5.1 clearly show an advantage to the F1-BCubed. Based on these results and recommendations form [3], we selected the F1-BCubed as our evaluation metric for our different clustering runs.

Now that we have selected a clustering evaluation metric, we can ran run our clustering experiments. To do so, we tested different pre-processing schemes: punctuation removal, stop-word removal and stemming.

Moreover, different feature extraction techniques were used: Term-Frequency (TF), Normalized Term-Frequency (L2-norm of the Term-frequency), TF-IDF and pre-trained fastText representations [1] with dimension 300. The fastText representation of a message was obtained by simply identifying the words of a chat message and averaging their word representations.

In addition to the aforementioned representations, we also decided to apply a simple Principle Component Analysis (PCA) in order to reduce the dimension of the 3 Bag-of-word representations we used: TF, normalized TF and TF-IDF. The representations we obtained had 300 (to match fastText vectors' size), 150 and 50 dimensions. Similarly, we applied a post-processing approach to the fastText vectors based on [80] to obtain fastText vectors of dimension 150 and 50 dimensions. The main objective of the dimension reduction is to make computations faster and scalable when applying them on larger data-sets. In total, we tested 15 different message representation approaches.

The reason we did not use more complex unsupervised vectorization approaches (supervised approaches do not necessarily work for all languages) is related to 3 reasons. First, more complex models like *skip-thoughts* are very slow to use and require a lot of computational resources. Moreover, some approaches have larger memory requirements like *sent2vec*. Finally, contrary to fastText, most other ways to obtain text vectors do not have a large repository of pre-trained models meaning that we would need to perform a new training each time we work with a new language.

When it comes to the clustering algorithms, we decided to limit our experiments to K-Means and HDBSCAN for now. We performed early experiments with other clustering algorithms however, they did not scale as well as the two aforementioned algorithms or, had parameters that were more delicate to tune such as DBSCAN's ϵ thus, making a fair comparison between different approaches difficult. Indeed, since we are dealing with annotated data, we provided K-Means with the real number of clusters K and HDBSCAN with the real *minClusterSize*. However, to determine the *minSamples* parameter of HDBSCAN, for one setting, we ran experiments using values of *minSamples* ranging from 1 to 2 * minClusterSize for the comparison to be fair. The idea here is to make sure that all the algorithms are provided with the same information about the data.

Regarding the distance metrics, we used the Euclidean distance and the cosine distance. For K-Means, we did not directly use the cosine distance (it uses the Euclidean distance

[1]Available at: https://fasttext.cc/

by default) rather, we normalized the vectors we fed to the algorithm to have a unit norm in order for the Euclidean distance to be equivalent to the cosine distance.

Once we identify the best approach using the 480 different combinations of parameters for each data-set (8 pre-processing approaches, 15 message representations, 2 algorithms and 2 distance metrics), we will select the approach that works best for most of the data-sets and replace the metric used by learning a Mahalanobis metric using the Mahalanobis Metric for Clustering [93]. The objective is to see if we can get some improvement. In this metric learning scheme we feed the algorithm with pairwise constraints telling the algorithm messages that should belong to the same class and messages that should not. Once the Mahalanobis matrix A is learned, we can use it to transform our data and apply a Euclidean distance on the transformed result.

5.3 Data

For our experiments, we used 5 of our annotated customer data-sets, 4 in Finnish and one in English. They were selected based on the customer importance and on the use-case. The data comes from different industries: telecommunications, airlines, postal services and e-commerce. For customer anonymity, we will refer to the different data-sets using their industry domain. The 2 telecommunications data-sets will be called telco1 and teclo2.

Data-set	Messages	Min length	Max length	Avg Length	Median Length
Telco1	8959	1 word	101 words	6.3 words	5 words
Airline	13258	1 word	157 words	14.75 words	11 words
Postal Services	2083	1 word	29 words	6.27 words	6 words
Telco2	1920	1 word	74 words	9.34 words	7 words
E-Commerce	3384	1 word	312 words	11.84 words	8 words

Table 5.2. *Message statistics of our different data-sets*

Table 5.2 contains information about the messages lengths of different data-sets. We can clearly see that we used varying size data-sets. The longest messages are to be found in Airline data-set. For all the data-sets, the shorter message was one word long, typically a greeting like "Hello". As mentioned earlier, although customer service messages tend to be relatively short on average, the absence of limitation in the message size creates cases where messages are more than 100 words long.

Table 5.3 contains statistics about the class annotation of our different data-sets. At first glance, we can easily see the class unbalance especially for the Airline data-set. In general, all of our data has a similar *minClusterSize* however, the largest cluster size can vary greatly. Another note is the fact that larger data-sets tend to have more classes.

Overall, we can see that our data contains a mix of messages with different sizes but,

Data-set	Classes	Min Class	Max Class	Avg Class	Median Class
Telco1	151	4 messages	286 messages	58.67 messages	39 messages
Airline	163	5 messages	1062 messages	79.89 messages	19.5 messages
Postal Services	57	6 messages	249 messages	36.54 messages	25 messages
Telco2	60	5 messages	78 messages	30.5 messages	26 messages
E-Commerce	42	6 messages	435 messages	75.27 messages	51 messages

Table 5.3. *Class statistics for our different data-sets*

are mostly short. In addition, our data-sets vary in size and content and are mostly unbalanced.

5.4 Results

For our experiments, we used a *c5.9xlarge* Amazon Web Service instance that comes with 72GB of RAM. The instance is shipped with an Intel Xeon Platinuim 8000 series with a Skylake architecture processor. It is equipped with an all core Turbo CPU clock speed of up to 3.5GHz.

As for the tools, all of our experiments were written in Python v3.6. We used UDpipe [87] that we trained on Universal Dependencies data [73] for message tokenization i.e spltting a message into words. We also took advantage of the Natural Language Toolkit (NLTK) [57] for the list of stopwords and the stemmer. In addition, we used Sklearn [76] for the K-Means implementation and Bag-of-words representations. The F1-BCubed was mostly based on python-bcubed's [43] implementation. Moreover, we used the HDBSCAN implementation available as part of scikit-learn contrib [64]. Finally metric-learn [65] was used for MMC experiments.

5.4.1 Hyper-Parameter Search

We will report the impact of the different parameters compared to a baseline that was used in the company before starting this work. The baseline consisted in using a fastText representation for messages with a K-Means algorithm (and Euclidean distance). The pre-processing step only involved the normalization of characters to lowercase which we will keep across all our experiments. In addition, all vlaues reported are F1-BCubed scores.

It is worth pointing out that, given the large parameter space, we could not afford to run our tests several times to account for the role of the K-Means centroid initialization hence, we fixed the seed of our random number generator to allow the repeatability of our results. In addition, in order to save space when reporting our experimental results, we decided to use abbreviations that can be found in Table 5.4.

Term	**Abbreviation**
Baseline	B
StopWord Removal	SWR
Punctuation Removal	PR
Vector Normalization	VN
Stemming	S
Term-Frequency	TF
Normalized Term-Frequency	NTF
Term-Frequency-Inverse-Document-Frequency	TF-IDF

Table 5.4. *List of abbreviations used when reporting experimental results*

Data-set	**B**	**B + SWR**	**B + SWR + PR**
Telco1	0.1141	0.1441	**0.1683**
Airline	0.1376	0.1481	**0.2038**
Postal Services	0.2631	0.3101	**0.3385**
Telco2	0.2127	0.2204	**0.2785**
E-Commerce	0.1889	0.2019	**0.2436**
Average	0.1833	0.2049	**0.2465**

Table 5.5. *The impact of stopword and punctuation removal on F1-BCubed scores*

Data-set	**B + SWR + PR**	**B + SWR + PR + VN**
Telco1	0.1683	**0.1808**
Airline	**0.2038**	0.1971
Postal Services	0.3385	**0.3578**
Telco2	0.2785	**0.3052**
E-Commerce	0.2436	**0.2440**
Average	0.2465	**0.2576**

Table 5.6. *Impact of vector normalization on the F1-BCubed scores*

Table 5.5 contains the results we obtained with our baseline together with the impact of removing stopwrods and punctuation. We can see that removing stop-words improves the quality of the clustering results for all the data-sets. The impact of the punctuation removal is even more visible particularly for the airline data-set which saw an improvement close to 50% compared to the baseline.

Following the success of the previous experiments, we wanted to study the impact of vector normalization on the results. Indeed, as we mentioned earlier, normalizing the input vectors to have a unit norm then, applying the Euclidean distance calculation is equivalent to applying a cosine distance calculation.

Table 5.6 contains the results we obtained when applying vector normalization on top of stopword removal and punctuation removal. Most of the data-sets benefit from a F1-BCubed improvement when applying vector normalization. The only exception is the Airline data-set which sees a slight degradation of its clustering results.

Building again on top of the previous improvement and based on the intuition that our data clusters might have different densities, we decided to replace K-Means with HDBSCAN. With the success of the vector normalization, we are incline to use the cosine distance with HDBSCAN. As mentioned earlier, since HDBSCAN has an additional parameter *minSamples*, we ran experiments with values of *minSamples* ranging from 1 to twice the *minClusterSize*. In all our experiments, for all the data-sets, the best results were obtained with a *minSamples* value of 1.

Data-set	**SWR+PR**	**B+SWR+PR+VN**	**HDBSCAN+cosine+SWR+PR**
Telco1	0.1683	0.1808	**0.3613**
Airline	0.2038	0.1971	**0.2176**
Postal Services	0.3385	0.3578	**0.4324**
Telco2	0.2785	0.3052	**0.3503**
E-Commerce	0.2436	0.2440	**0.3664**
Average	0.2465	0.2576	**0.3456**

Table 5.7. *F1-BCubed scores using HDBSCAN and cosine distance instead of K-Means*

Results in Table 5.7 contain F1-BCubed scores obtained when using HDBSCAN with cosine distance as a replacement for K-Means. We can see a sizable improvement in clustering quality for all the data-sets. This is particulary visible for Telco1 which saw its score double compared to the previous experimental setting.

In an effort to further improve our results, we decided to test the different feature extraction techniques we described earlier. Please note that, full-dimension TF, normalized TF and TF-IDF did not scale very well to the Airline data-set. After further inspection, it turns out that the dimension of the bag-of-words representation was fairly large as this can be seen in Table 5.8 where we can clearly see that the BoW dimension of the Airline data-set if way larger than that of other data-sets. Based on this, we decided to discard non reduced BoW representations as they would probably not scale to our larger unannotated data-sets.

Data-set	Bag-of-words dimension
Telco1	8392
Airline	24100
Postal Services	2274
Telco2	3435
E-Commerce	2802

Table 5.8. *Dimension of the Bag-of-words representation for our different data-sets*

Data-set	TF300	TF150	TF50	NTF300	NTF150	NTF50	TFIDF300	TFIDF150	TFIDF50	FastText300	FastText150	FastText50
Telco1	0.2353	0.2171	0.1924	0.2393	0.2222	0.2041	0.2531	0.2338	0.222	0.3613	**0.3621**	0.3420
Airline	0.2671	0.2543	0.2268	0.2745	0.2637	0.2264	**0.2893**	0.2797	0.254	0.2176	0.2357	0.1404
Postal Services	0.3671	0.3303	0.2643	0.3639	0.3369	0.2571	0.365	0.3413	0.3142	**0.4324**	0.407	0.3983
Telco2	0.3137	0.2859	0.232	0.3226	0.2912	0.2447	0.3368	0.2963	0.2756	**0.3503**	0.3356	0.2911
E-Commerce	0.3628	0.3418	0.2978	**0.3706**	0.3543	0.3008	0.3382	0.3282	0.3126	0.3664	0.3022	0.2782
Average	0.3092	0.2859	0.2427	0.3142	0.2937	0.2466	0.3165	0.2959	0.2757	**0.3456**	0.3285	0.2900

Table 5.9. *F1-BCubed scores obtained with HDBSCAN and cosine distance for several message representation schemes*

Data-set	TF300	TF150	TF50	NTF300	NTF150	NTF50	TFIDF300	TFIDF150	TFIDF50	FastText300	FastText150	FastText50
Telco1	0.2794	0.2447	0.2067	0.2848	0.2428	0.2185	0.2884	0.2606	0.2347	**0.3652**	0.3637	0.3278
Airline	0.2808	0.2778	0.254	0.287	0.2816	0.2534	**0.3084**	0.2932	0.2826	0.2181	0.222	0.1529
Postal Services	0.385	0.3759	0.29	0.3929	0.3818	0.2876	0.3883	0.3808	0.3425	**0.3955**	0.3859	0.3362
Telco2	0.3372	0.3077	0.2624	0.3485	0.3221	0.2815	**0.3579**	0.3387	0.2989	0.321	0.3306	0.2410
E-Commerce	0.3596	0.3446	0.2955	0.3618	**0.3641**	0.3049	0.3353	0.3227	0.3046	0.3535	0.2951	0.3191
Average	0.3284	0.3101	0.2617	0.3350	0.3185	0.2692	**0.3357**	0.3192	0.2927	0.3307	0.3195	0.2754

Table 5.10. *F1-BCubed scores obtained with HDBSCAN and cosine distance for several message representation schemes when applying stemming*

Table 5.9 contains results obtained using the HDBSCAN clustering algorithm with cosine distance, stopword and punctuation removal for several message representation approaches. The numbers appended to the message representation schemes reflect the dimension of the representation. We can see that, in general, within the same family, representations having less dimensions tend to perform worse than those with more dimensions. In addition, the fasText representation with 300 dimensions seems to be the most competitive. The Airline data-set saw a large improvement using a TFIDF300 approach. As a matter of fact, this is the only data-set where the fastText representation does not compare well to the bag-of-words approaches.

Table 5.10 contains results we obtained using the same experimental setting as in Table 5.9 except that we additionally apply stemming as a pre-processing step- From the results we notice again that, for one given message representation scheme, higher dimensions tend to perform better. The TFIDF300 representation seems to work better when stemming is applied. Overall, there was not a big change in terms of results except for the Postal services data-set that saw a noticeable drop in performance and the Airlines data-set that saw an fair increase clustering quality.

Given that TFIDF300 and fastText300 seemed to be strong candidates when it comes to message representation, we decided to have a look at their results more closely and see how stemming impacts them. Table 5.11 contains the condensed results for the two message representation approaches. It seems that the fastText representation suffers slightly when applying stemming as a pre-processing step. Conversly, the TFIDF approach seems to work much better when using stemming. As for the comparison between the two approaches, there does not seem to be a clear edge for either of the methods.

Data-set	**TFIDF300**	**FastText300**	**TFIDF300 + S**	**FasText300 + S**
Telco1	0.2531	0.3613	0.2884	**0.3652**
Airline	0.2893	0.2176	**0.3084**	0.2181
Postal Services	0.365	**0.4324**	0.3883	0.3955
Telco2	0.3368	0.3503	**0.3579**	0.3210
E-Commerce	0.3382	**0.3664**	0.3353	0.3535
Average	0.3165	**0.3456**	0.3357	0.3307

Table 5.11. *Close comparison between F1-BCubed scores obtained for TFIDF300 and fastText300 vis-à-vis the usage of stemming*

Before moving on to the next part of our experiments: Mahalanobis metric learning, we decided to choose the TFIDF300 representation as our base representation for metric learning (more on this in the discussion part). Table 5.12 contains a summary of the improvements we obtained when tuning different clustering parameters.

Data-set	**Baseline**	**New Approach**
Telco1	0.1141	**0.2884**
Airline	0.1376	**0.3084**
Postal Services	0.2631	**0.3883**
Telco2	0.2127	**0.3579**
E-Commerce	0.1889	**0.3353**
Average	0.1883	**0.3357**

Table 5.12. *Comparison between F1-BCubed results obtained with the baseline and results obtained after tuning different clustering parameters*

5.4.2 Metric Learning

For the Mahalanobis Metric for Clustering (MMC), we create training data consisting in pairs of messages with a label 0 if they do not belong to the same class and 1 if they belong to the same class. In addition, since the number of possible samples that are from different classes is usually far bigger than the number of samples coming from the same class, we decided to remove to surplus of negative examples so that positive and negative examples are equal i.e we use a balanced training set.

For our experimental setting, we had two alternatives. The first alternative is to split a customer data-set in half in a stratified way so that the class distribution is the same across both halves. The two resulting halves are labelled training and test. First, we cluster the test part using the parameters we found in the previous subsection. Afterwards, we use the training part to learn a metric then, we evaluate and run the clustering again using the metric on the test part. In this first alternative, one needs to keep in mind that the numbers reported cannot be related to those we saw in the previous subsection as we are only evaluating on half of the data.

The second approach consisted in taking our 4 Finnish data-sets (the language is not important as long as they are from the same language) and learn a metric using one customer data-set and test the learned metric's performance on the other bots.

Data-set	**Base Score**	**MMC Score**
Telco1	0.2959	**0.3575**
Airline	0.3083	**0.3107**
Postal Services	0.3949	**0.3980**
Telco2	0.3473	**0.3687**
E-Commerce	0.3603	**0.3910**

Table 5.13. *F1-BCubed scores obtained on 50% of the data before and after MMC metric learning*

Table 5.13 contains the results obtained when learning the metric on half a customer's data and testing on the other half i.e the first alternative. We can see that all the bots benefited from the metric learning using MMC, more particulary, Telco1 and E-Commerce data-sets.

Baseline	**0.2884**	**0.3084**	**0.3883**	**0.3579**
Trained On \Trained for	**Telco1**	**Airline**	**Postal Services**	**Telco2**
Telco1	NA	0.2608	0.3042	0.2651
Airline	0.2198	NA	0.2667	0.2153
Postal Services	0.2478	0.2300	NA	0.2142
Telco2	0.2202	0.2317	0.2523	NA

Table 5.14. *F1-BCubed obtained when training a metric on one data-set and using it for another data-set*

Table 5.14 contains results obtained with the second alternative using MMC. In this case, the scores can be compared to the scores obtained in the previous subsection as we evaluate on the whole data-set. We clearly see that no pair of training increased the clustering score. In addition, we notice the most performing data-set when it comes to learning the metric is Telco1. As for the potential benefit of training on a bot from the same industry, we can see that Telco1 did not benefit to Telco2 and vice-versa.

5.5 Discussion

Data pre-processing in general and stopword and punctuation removal in particular can have a huge impact on clustering quality as we have seen in the results from Table 5.5. This is not surprising as their presence dilutes the representation power of the different feature extraction techniques. Indeed, if we think about the fastText representation which is the result of averaging the vectors of the words present in the sentence, the more vectors there are, the less important vectors will have the chance to have their say in the end result. For instance, a sentence like "the case of the password?" which has 3 stop words (the, the, of) and 1 punctuation mark (?) will be hard to represent if we do not get rid of the stopwords as the average will tend to move closer to the vector of "the".

This reasoning can be extended to explain why the Postal Services data-set had the highest numerical score in different experiments while the Airline data-set has the lowest score. Indeed, if we look at the message length statistics for the Postal data, we can see that, on average it contains the shortest chat messages. Conversly, the airline data-set contains the longest messages. Again, averaging long messages produces less useful vectors. This is why the Airline data-set performed better when using a TF-IDF300 representation. Given the way they are obtained, TFIDF vectors seem to be better suited for representing longer chunks of text.

When it comes to vector normalization, we saw an improvement in the quality of our clustering result thus suggesting that the use of the cosine distance was, in general, better for our task. This was confirmed by our extended experiments where the cosine distance largely outperformed the Euclidean distance. This is probably due to the fact that the cosine distance ignores the scale of the different dimensions which might bias the results.

Our experiments also showed that using HDBSCAN significantly boosted the clustering performance compared to using K-Means. This seems reasonable as K-Means will try to fit every cluster point around a centroid which is not always suitable as we have seen in Figure 4.2. In addition, K-Means is highly sensitive to outliers and does not have a notion of noise points. When looking at the actual clustering results, we could clearly see that K-Means produced large clusters while HDBSCAN produced several smaller clusters and 1 big noise cluster containing data with a lot of everything: actual noise (like messages containing one letter only), useful messages and complicated messages dealing with several topics. Nevertheless, the clusters that HDBSCAN did not consider as noise were clustered in a generally correct and neat way. This means that there is a lot of potential when it comes to having a starting point for data annotation. In addition, it offered the advantage of being easier to process as we only care about the non-noisy points.

Before making the switch to HDBSCAN, we decided to ask all 3 members of the annotation team whether they felt it was easier to select the number of clusters (K-Means) or the minimum cluster size (HDBSCAN). The results we obtained are in the Table 5.15. We can clearly see that choosing the *minClusterSize* does not seem to be a problem.

	minClusterSize	**Number of Clusters**	**Neutral**
Votes	**1**	**0**	**2**

Table 5.15. *Annotation team's view regarding clustering parameter choice*

Moving forward to the impact of stemming. We could see that TFIDF benefited more from stemming than the fastText representation. This is probably due to the fact that stemming reduces the dimension of the original TF-IDF vectors significantly. As a matter of fact, in the case of the Airline data-set went from 24100 dimensions to 15892 dimensions i.e almost a 40% decrease. Given that we apply PCA on top of TF-IDF, the chance of having more meaningful dimensions in the end result becomes higher.

When it comes to our choice of message representation scheme, we decided to go with the TF-IDF300 even if the fastText approach was competitive based on our results. The first motivation for this choice is the one we mentioned above regarding the ability of the TF-IDF approach to cope with longer messages. This seems to be relevant as many of our unannotated data-sets seem to have message lengths that are closer to those of the Airline data-set than of the Postal services which worked well with fastText Representations. Table 5.16 contains statistics about some of our unannotated data.

The second reason we selected the TFIDF300 was related to the fact that clustering results using this approach were easy to interpret. In other words, one can clearly see that one cluster revolves around a word and its derivatives. For instance, you could look at 10 to 15 messages of one cluster to identify that most messages contain terms like

Data-set	Messages	Min Length	Max Length	Avg Length	Median Length
Customer A	34260	1	235	12.74	8
Customer B	96202	1	176	11.85	9
Customer C	177132	1	1646	12.05	9

Table 5.16. *Message statistics regarding some of our unannotated data-sets*

"responsible" and "responsibility". This ease of interpretation means that we can easily ignore a cluster if we think that its content is not suitable without having to go through all of its messages. In addition, it helps name the different classes that are produced from the clustering results: "Shopping", "Legal Responsibility" etc.

When it comes to the metric learning results, we experimented with two possible ways of training our metric for improving clustering results. The option where we trained and evaluated on one customer data-set showed promising results. The failure of the second alternative (training on one data-set for another) is probably due to the fact that topics and products that customers deal with when interacting with a company cannot necessarily be transferred to another company since, different companies have different issues requiring customers to ask different questions even though the two companies work in the same industry.

Our initial clustering results on real unannotated data-sets using only the newly found clustering parameters are promising. Based on our metric learning experiments and to improve our clustering results even further, we propose the following clustering procedure:

1. Randomly select a subset of a customer's unannotated data messages that can be processed in a reasonable amount of time (ex: 10K messages),
2. Run clustering on the selected data and extract classes from it,
3. Based on the annotation obtained, apply the MMC for learning a Mahalanobis matrix,
4. Use the newly learned matrix to cluster the rest of the customer data,
5. Repeat 1-4 if necessary

Such an approach can be very useful for very large data-sets. In addition, the metric learning scheme can be used directly from step 3 when the objective of clustering is to assign newly arrived messages to an already existing data-set. One could imagine training a metric on the whole Airline data-set upon reception of a new batch of customer messages.

5.6 Conclusion

In this chapter, we described our approach for tackling the problem of clustering customer service chat messages. We described the data-sets we used for our experiments and reported the different results we obtained. Then, we discussed our results and tried to explain why certain approaches behaved the way they did. Finally, we proposed a procedure for taking advantage of metric learning for clustering our customer messages.

6 GENERAL CONCLUSION AND FUTURE WORK

In this work, we dealt the problem of clustering customer service chat messages with the objective of providing a language agnostic, industry independent, easy to use and scalable approach for simplifying and speeding up the topic annotation process of chat messages at ultimate.ai.

Our contribution covers 3 aspects. Firstly, we were able to identify an evaluation metric that is suitable for ultimate.ai's customer service chat data. This allowed the development of an experimental set-up within the company that allows the future development of clustering approaches. Secondly, we were able to significantly improve the clustering results compared to the baseline previously in-use at the company. Thridly, we proposed a clustering setting allowing the use of a metric learning scheme for further clustering improvement.

Customer service chat messages are different from most of the textual data being dealt with in the clustering literature in two aspects. First, customer service chat data is highly colloquial meaning that it contains a lot of spoken language and spelling mistakes. Moreover, while most customer messages are short, a sizable proportion of our data spans multiple sentences.

While clustering involves a lot moving pieces, in our work, we looked into 5 major aspects. First, we looked into different ways of pre-processing our data by studying the impact of stopword removal, punctuation removal and stemming.

Moreover, we covered different ways of numerically representing our chat messages and extracting useful features from them while keeping in mind the scalability and generality of the different techniques. Our results show that the performance of some message representation approaches have a lot to do with the message length they deal with.

In addition, we studied few clustering algorithms by looking into the way they handle noise points and the type of clusters they produced. We also payed attention to the ease of parameter selection and scalability of the algorithms while taking into account the format of the results produced as it impacts the result processing speed.

Finally, we dived into the issue of clustering evaluation which is a difficult task. Although clustering evaluation is still an on-going research topic, we were able to identify and use a similarity metric that works well with our experimental setting and data.

When it comes to future research directions, we would like to validate the quality of our evaluation metric even further by having more extensive qualitative assessments. We also would like to look more into data pre-processing. More specifically, we want to study the impact of a spell-correction system on our results. Another data pre-proessing scheme involves lemmatization whereby we would obtain the grammatical roots of words.

Furthermore, we believe that looking into different dimension-reduction techniques such as UMAP [63] will prove to be important not only from the clustering perspective but also form the data visualization aspect as it will provide additional tools for data understanding and annotation and provide business value for customers who want visual insight on their data.

Finally, a deeper look into different metric learning algorithms could help our current approaches to perform better while making use of the currently available data and tools. Indeed, our current work only scratches the surface of what can be achieved through different metric learning schemes.

REFERENCES

[1] O. Abiola, A. Adetunmbi, and A. Oguntimilehin. Using hybrid approach for english-to-yoruba text to text machine translation system (proposed). In: *International Journal of Computer Science and Mobile Computing* 4.8 (2015), 308–313.

[2] C. C. Aggarwal and C. Zhai. A survey of text clustering algorithms. In: *Mining text data*. Springer, 2012, 77–128.

[3] E. Amigó, J. Gonzalo, J. Artiles, and F. Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. In: *Information retrieval* 12.4 (2009), 461–486.

[4] S. Arora, Y. Liang, and T. Ma. A simple but tough-to-beat baseline for sentence embeddings. In: (2016).

[5] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In: *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics-Volume 1*. Association for Computational Linguistics. 1998, 79–85.

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. In: *arXiv preprint arXiv:1409.0473* (2014).

[7] S. Banerjee, K. Ramanathan, and A. Gupta. Clustering short texts using wikipedia. In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM. 2007, 787–788.

[8] M. Bates and R. M. Weischedel. *Challenges in natural language processing*. Cambridge University Press, 2006.

[9] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. In: *arXiv preprint arXiv:1306.6709* (2013).

[10] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. In: *Journal of computational biology* 6.3-4 (1999), 281–297.

[11] E. Boiy and M.-F. Moens. A machine learning approach to sentiment analysis in multilingual Web texts. In: *Information retrieval* 12.5 (2009), 526–558.

[12] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov. Enriching word vectors with subword information. In: *arXiv preprint arXiv:1607.04606* (2016).

[13] S. R. Bowman, G. Angeli, C. Potts, and C. D. Manning. A large annotated corpus for learning natural language inference. In: *arXiv preprint arXiv:1508.05326* (2015).

[14] P. F. Brown, V. J. D. Pietra, S. A. D. Pietra, and R. L. Mercer. The mathematics of statistical machine translation: Parameter estimation. In: *Computational linguistics* 19.2 (1993), 263–311.

[15] R. J. Campello, D. Moulavi, and J. Sander. Density-based clustering based on hierarchical density estimates. In: *Pacific-Asia conference on knowledge discovery and data mining*. Springer. 2013, 160–172.

[16] D. Cer, Y. Yang, S.-y. Kong, N. Hua, N. Limtiaco, R. S. John, N. Constant, M. Guajardo-Cespedes, S. Yuan, C. Tar, et al. Universal sentence encoder. In: *arXiv preprint arXiv:1803.11175* (2018).

[17] O. Chapelle, B. Scholkopf, and A. Zien. Semi-supervised learning (chapelle, o. et al., eds.; 2006)[book reviews]. In: *IEEE Transactions on Neural Networks* 20.3 (2009), 542–542.

[18] D. Chen and C. Manning. A fast and accurate dependency parser using neural networks. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, 740–750.

[19] M. Chen, Y. Hao, K. Hwang, L. Wang, and L. Wang. Disease prediction by machine learning over big data from healthcare communities. In: *IEEE Access* 5 (2017), 8869–8879.

[20] W. Chen and M. Zhang. *Semi-Supervised Dependency Parsing*. Springer, 2015.

[21] G. G. Chowdhury. Natural language processing. In: *Annual review of information science and technology* 37.1 (2003), 51–89.

[22] A. Clark, C. Fox, and S. Lappin. *The handbook of computational linguistics and natural language processing*. John Wiley & Sons, 2013.

[23] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes. Supervised learning of universal sentence representations from natural language inference data. In: *arXiv preprint arXiv:1705.02364* (2017).

[24] G. Cybenko. Approximation by superpositions of a sigmoidal function. In: *Mathematics of control, signals and systems* 2.4 (1989), 303–314.

[25] *Data Never Sleeps 6 | Domo*. June 2018. URL: `https://www.domo.com/learn/data-never-sleeps-6#/`.

[26] B. De Gelder. Towards the neurobiology of emotional body language. In: *Nature Reviews Neuroscience* 7.3 (2006), 242.

[27] P. Domingos. A few useful things to know about machine learning. In: *Communications of the ACM* 55.10 (2012), 78–87.

[28] C. Donalek. Supervised and unsupervised learning. In: *Astronomy Colloquia. USA*. 2011.

[29] I. El Naqa and M. J. Murphy. What is machine learning? In: *Machine Learning in Radiation Oncology*. Springer, 2015, 3–11.

[30] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. Density-based spatial clustering of applications with noise. In: *Int. Conf. Knowledge Discovery and Data Mining*. Vol. 240. 1996.

[31] M. Faruqui, Y. Tsvetkov, P. Rastogi, and C. Dyer. Problems with evaluation of word embeddings using word similarity tasks. In: *arXiv preprint arXiv:1605.02276* (2016).

[32] S. Feldman. NLP meets the Jabberwocky: Natural language processing in information retrieval. In: *ONLINE-WESTON THEN WILTON-* 23 (1999), 62–73.

[33] S. Fodeh, B. Punch, and P.-N. Tan. On ontology-driven document clustering using core semantic features. In: *Knowledge and information systems* 28.2 (2011), 395–421.

[34] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with LSTM. In: (1999).

[35] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In: *Proceedings of the fourteenth international conference on artificial intelligence and statistics*. 2011, 315–323.

[36] Y. Goldberg. Neural network methods for natural language processing. In: *Synthesis Lectures on Human Language Technologies* 10.1 (2017), 1–309.

[37] Y. Goldberg and O. Levy. word2vec Explained: deriving Mikolov et al.'s negative-sampling word-embedding method. In: *arXiv preprint arXiv:1402.3722* (2014).

[38] N. Gupta and P. Mathur. Spell checking techniques in NLP: a survey. In: *International Journal of Advanced Research in Computer Science and Software Engineering* 2.12 (2012).

[39] J. Han, J. Pei, and M. Kamber. *Data mining: concepts and techniques*. Elsevier, 2011.

[40] Z. S. Harris. Distributional structure. In: *Word* 10.2-3 (1954), 146–162.

[41] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. In: *Neural networks* 2.5 (1989), 359–366.

[42] *How many languages are there in the world?* Aug. 2018. URL: `https://www.ethnologue.com/guides/how-many-languages`.

[43] H. Hromic. *Python-bcubed*. `https://github.com/hhromic/python-bcubed`. 2018.

[44] X. Hu, J. Tang, H. Gao, and H. Liu. Unsupervised sentiment analysis with emotional signals. In: *Proceedings of the 22nd international conference on World Wide Web*. ACM. 2013, 607–618.

[45] L. Hubert and P. Arabie. Comparing partitions. In: *Journal of classification* 2.1 (1985), 193–218.

[46] S. S. Im Walde. Experiments on the automatic induction of German semantic verb classes. In: *Computational Linguistics* 32.2 (2006), 159–194.

[47] A. K. Jain. Data clustering: 50 years beyond K-means. In: *Pattern recognition letters* 31.8 (2010), 651–666.

[48] A. K. Jain and R. C. Dubes. Algorithms for clustering data. In: (1988).

[49] D. Jurafsky. *Minimum Edit Distance*. URL: `https://web.stanford.edu/class/cs124/lec/med.pdf`.

[50] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In: *Advances in neural information processing systems*. 2015, 3294–3302.

[51] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*. 2012, 1097–1105.

[52] D. Kuang, J. Choo, and H. Park. Nonnegative matrix factorization for interactive topic modeling and document clustering. In: *Partitional Clustering Algorithms*. Springer, 2015, 215–243.

[53] K. Leung and C. Leckie. Unsupervised anomaly detection in network intrusion detection using clusters. In: *Proceedings of the Twenty-eighth Australasian conference on Computer Science-Volume 38*. Australian Computer Society, Inc. 2005, 333–342.

[54] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In: *Soviet physics doklady*. Vol. 10. 8. 1966, 707–710.

[55] O. Levy and Y. Goldberg. Neural word embedding as implicit matrix factorization. In: *Advances in neural information processing systems*. 2014, 2177–2185.

[56] E. D. Liddy. Enhanced text retrieval using natural language processing. In: *Bulletin of the American Society for Information Science and Technology* 24.4 (1998), 14–16.

[57] E. Loper and S. Bird. NLTK: The Natural Language Toolkit. In: *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*. 2002.

[58] A. A. Lunsford and K. J. Lunsford. " Mistakes are a fact of life": A national comparative study. In: *College Composition and Communication* (2008), 781–806.

[59] M.-T. Luong, H. Pham, and C. D. Manning. Effective approaches to attention-based neural machine translation. In: *arXiv preprint arXiv:1508.04025* (2015).

[60] J. MacQueen et al. Some methods for classification and analysis of multivariate observations. In: *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*. Vol. 1. 14. Oakland, CA, USA. 1967, 281–297.

[61] C. D. Manning, C. D. Manning, and H. Schütze. *Foundations of statistical natural language processing*. MIT press, 1999.

[62] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: The Penn Treebank. In: *Computational linguistics* 19.2 (1993), 313–330.

[63] L. McInnes and J. Healy. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. In: *ArXiv e-prints* (Feb. 2018). arXiv: `1802.03426 [stat.ML]`.

[64] L. McInnes, J. Healy, and S. Astels. hdbscan: Hierarchical density based clustering. In: *The Journal of Open Source Software* 2.11 (2017), 205.

[65] metric-learn. *metric-learn*. `https://github.com/metric-learn`. 2018.

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

[67] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In: *Advances in neural information processing systems*. 2013, 3111–3119.

[68] T. M. Mitchell et al. Machine learning. 1997. In: *Burr Ridge, IL: McGraw Hill* 45.37 (1997), 870–877.

[69] D. Moulavi, P. A. Jaskowiak, R. J. Campello, A. Zimek, and J. Sander. Density-based clustering validation. In: *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM. 2014, 839–847.

[70] N. K. Nagwani and A. Sharaff. SMS spam filtering and thread identification using bi-level text classification and clustering techniques. In: *Journal of Information Science* 43.1 (2017), 75–87.

[71] J. L. Neto, A. D. Santos, C. A. Kaestner, N. Alexandre, D. Santos, et al. Document clustering and text summarization. In: (2000).

[72] J. Nivre. *Inductive dependency parsing*. Springer, 2006.

[73] J. Nivre, M.-C. De Marneffe, F. Ginter, Y. Goldberg, J. Hajic, C. D. Manning, R. T. McDonald, S. Petrov, S. Pyysalo, N. Silveira, et al. Universal Dependencies v1: A Multilingual Treebank Collection. In: *LREC*. 2016.

[74] J. Oladosu, A. Esan, I. Adeyanju, B. Adegoke, O. Olaniyan, and B. Omodunbi. Approaches to machine translation: a review. In: *FUOYE Journal of Engineering and Technology* 1.1 (2016).

[75] M. Pagliardini, P. Gupta, and M. Jaggi. Unsupervised learning of sentence embeddings using compositional n-gram features. In: *arXiv preprint arXiv:1703.02507* (2017).

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

[77] J. Pennington, R. Socher, and C. Manning. Glove: Global vectors for word representation. In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, 1532–1543.

[78] J. L. Peterson. Computer programs for detecting and correcting spelling errors. In: *Communications of the ACM* 23.12 (1980), 676–687.

[79] G. Petkos, S. Papadopoulos, and Y. Kompatsiaris. Two-level Message Clustering for Topic Detection in Twitter. In: *SNOW-DC@ WWW*. 2014, 49–56.

[80] V. Raunak. Effective Dimensionality Reduction for Word Embeddings. In: *arXiv preprint arXiv:1708.03629* (2017).

[81] S. Romano, N. X. Vinh, J. Bailey, and K. Verspoor. Adjusting for chance clustering comparison measures. In: *The Journal of Machine Learning Research* 17.1 (2016), 4635–4666.

[82] P. J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. In: *Journal of computational and applied mathematics* 20 (1987), 53–65.

[83] J. M. Santos and M. Embrechts. On the use of the adjusted rand index as a metric for evaluating supervised classification. In: *International Conference on Artificial Neural Networks*. Springer. 2009, 175–184.

[84] Y. Shen, Z. Lin, A. P. Jacob, A. Sordoni, A. Courville, and Y. Bengio. Straight to the Tree: Constituency Parsing with Neural Syntactic Distance. In: *arXiv preprint arXiv:1806.04168* (2018).

[85] V. Sindhwani and P. Melville. Document-word co-regularization for semi-supervised sentiment analysis. In: *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. IEEE. 2008, 1025–1030.

[86] M. Steinbach, G. Karypis, V. Kumar, et al. A comparison of document clustering techniques. In: *KDD workshop on text mining*. Vol. 400. 1. Boston. 2000, 525–526.

[87] M. Straka and J. Straková. Tokenizing, POS Tagging, Lemmatizing and Parsing UD 2.0 with UDPipe. In: *Proceedings of the CoNLL 2017 Shared Task: Multilingual Parsing from Raw Text to Universal Dependencies*. Vancouver, Canada: Association for Computational Linguistics, Aug. 2017, 88–99. URL: `http://www.aclweb.org/anthology/K/K17/K17-3009.pdf`.

[88] A. Taylor, M. Marcus, and B. Santorini. The Penn treebank: an overview. In: *Treebanks*. Springer, 2003, 5–22.

[89] L. E. Thorelli. Automatic correction of errors in text. In: *BIT Numerical Mathematics* 2.1 (1962), 45–52.

[90] M. Tomita. Sentence disambiguation by asking. In: *Efficient Parsing for Natural Language*. Springer, 1986, 103–119.

[91] N. X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. In: *Journal of Machine Learning Research* 11.Oct (2010), 2837–2854.

[92] F. Wang and J. Sun. Survey on distance metric learning and dimensionality reduction in data mining. In: *Data Mining and Knowledge Discovery* 29.2 (2015), 534–564.

[93] E. P. Xing, M. I. Jordan, S. J. Russell, and A. Y. Ng. Distance metric learning with application to clustering with side-information. In: *Advances in neural information processing systems*. 2003, 521–528.

[94] J. Xu, B. Xu, P. Wang, S. Zheng, G. Tian, and J. Zhao. Self-taught convolutional neural networks for short text clustering. In: *Neural Networks* 88 (2017), 22–31.

[95] M. Zareapoor and P. Shamsolmoali. Application of credit card fraud detection: Based on bagging ensemble classifier. In: *Procedia Computer Science* 48 (2015), 679–685.