Advancing Keyword Clustering Techniques: A Comparative Exploration of Supervised and Unsupervised Method

Tesi di Laurea Magistrale in
Computer Science - Ingegneria Informatica

Author: **Filippo Caliò**

Student ID: 994184
Advisor: Prof. Davide Martinenghi
Co-advisors: Sarunas, Girdzijauskas
Academic Year: 2022-23

Abstract

Clustering keywords is an important Natural Language Processing task that can be adopted by several businesses since it helps to organize and group related keywords together. By clustering keywords, businesses can better understand the topics their customers are interested in. This thesis project provides a detailed comparison of two different approaches that might be used for performing this task and aims to investigate whether having the labels associated with the keywords improves the clusters obtained. The keywords are clustered using both supervised learning, training a neural network and applying community detection algorithms such as Louvain, and unsupervised learning algorithms, such as HDBSCAN and K-Means. The evaluation is mainly based on metrics like NMI and ARI.

The results show that supervised learning can produce better clusters than unsupervised learning. By looking at the NMI score, the supervised learning approach composed by training a neural network with Margin Ranking Loss and applying Kruskal achieves a slightly better score of 0.771 against the 0.693 of the unsupervised learning approach proposed, but by looking at the ARI score, the difference is more relevant. HDBSCAN achieves a lower score of 0.112 compared to the supervised learning approach with the Margin Ranking Loss (0.296), meaning that the clusters formed by HDBSCAN may lack meaningful structure or exhibit randomness.

According to the evaluation metrics, the study reveals that the supervised learning approach with the Margin Ranking Loss creates more accurate clusters than the unsupervised learning techniques, but training with a BCE loss function provides different results, obtaining that the unsupervised algorithms are better than this latter supervised learning approach.

Keywords: Keyword Clustering, Supervised Learning, Unsupervised Learning, Cluster Labels, Natural Language Processing, Sentence Embeddings

Abstract in lingua italiana

Il clustering delle parole chiave è un'importante ramo di Natural Language Processing che può essere adottata da diverse aziende, poiché aiuta a organizzare e raggruppare diverse parole chiave. Il clustering delle parole chiave consente alle aziende di comprendere meglio gli argomenti a cui sono interessati i loro clienti. Questo progetto di tesi fornisce un confronto dettagliato di due diversi approcci che potrebbero essere utilizzati per svolgere questo compito e mira a indagare se la presenza di etichette associate alle parole chiave migliora i cluster ottenuti. Le parole chiave vengono raggruppate utilizzando sia supervised learning, addestrando una rete neurale e applicando algoritmi di community detection come Louvain, sia algoritmi di unsupervised learning, come HDBSCAN e K-Means. La valutazione si basa principalmente su metriche come NMI e ARI.

I risultati mostrano che il supervised learning può produrre cluster migliori rispetto all'unsupervised learning. Osservando il punteggio NMI, l'approccio di supervised learning composto dal training di una rete neurale con Margin Ranking Loss e l'applicazione di Kruskal raggiunge un punteggio leggermente migliore di 0,771 contro lo 0,693 dell'approccio di unsupervised learning proposto, ma osservando il punteggio ARI, la differenza è più rilevante. HDBSCAN ottiene un punteggio inferiore di 0,112 rispetto all'approccio di supervised learning con Margin Ranking Loss (0,296), il che significa che i cluster formati da HDBSCAN potrebbero mancare di una struttura significativa o presentare una certa casualità.

In base alle metriche di valutazione, lo studio rivela che l'approccio di supervised learning con Margin Ranking Loss crea cluster più accurati rispetto alle tecniche di unsupervised learning, ma l'addestramento con BCE come loss function fornisce risultati diversi, ottenendo che gli algoritmi unsupervised sono migliori di quest'ultimo approccio di supervised learning.

Parole chiave: Clustering di parole chiave, Apprendimento Supervisionato, Apprendimento Non Supervisionato, Etichette dei cluster, Natural Language Processing

Contents

1 | Introduction

Today, there is an inevitable increase in the amount of data that has become available online and comes from different sources [4]. They have the potential to be helpful to various kind of businesses and can be used to create value. However, in order to do this, it is required to work with this data, extracting useful information. One of the most widely used techniques is clustering [49], which is the process of grouping similar objects together, based on their similarity, such that items in the same group are more similar to each other than items in other groups [49, 77]. Clustering algorithms are becoming important tools for analyzing data, especially to separate large amount of data into smaller groups. In today's data-driven world, the ability to automatically group data into clusters has become critical in several areas, including recommendation systems, search engine optimization (SEO) and social networks [44]. Organizing and categorizing data, for example text, is crucial in extracting useful information that can help businesses to make decisions. By grouping text, companies and organizations can better understand their data, find out emerging trends and topics, and improve their recommendation engine. By comprehending the context and themes connected to various types of keywords, they can make better recommendations to users.

Additionally, this procedure may lead to further applications, like sentiment analysis [72] and topic modelling [8]. The former is a technique used to determine the emotional tone expressed in a piece of text, for example a review or a customer feedback. The latter involves identifying and extracting topics or themes from a collection of documents or text data.

A variety of community detection algorithms can be used to cluster keywords effectively [26, 46]. They can be applied to keyword clustering by treating words as nodes in a graph and examining the similarity relationships among them. Community detection algorithms can help identify word groups (communities) based on their semantic meaning.

However, traditional community detection algorithms may not always yield optimal results for keyword clustering as they are unsupervised and do not consider external information or domain-specific knowledge. This is where supervised clustering algorithms may be useful [33, 34]. By incorporating additional information such as labeled data, these algorithms may be able to increase the accuracy of the clusters. In supervised clustering, the algorithm exploits a predetermined set of categories or labels, associated to the data. The clustering process is then trained and optimized by the algorithm using these labels [33, 34].

In this thesis, we propose combining the supervised clustering algorithm with several community detection algorithms such as Louvain [11], one of the most used algorithms for weighted graphs, to cluster keywords. We aim to find out how well community detection algorithms work for clustering keywords and if using supervised clustering algorithms make clusters more distinct and clear than the ones created by unsupervised techniques [30], such as HDBSCAN and K-Means. The aim is to find out whether having or not labels associated to text data is helpful to cluster keywords.

1.1. Background

Natural Language Processing (NLP) [15] is the name given to the area of computer science that helps interpret text, and more specifically, the area of Artificial Intelligence (AI) and deep learning. This aids in the understanding and processing of text that contains human language by computers. It is used to translate text between languages [43], responds to text commands, and so on. It is now being used more frequently than ever [15] because there are now larger volumes of textual data available and also because machine learning techniques like word embeddings are being used [8]. One of the challenges regarding NLP is representing text in such a way that machines can understand [40]. During the last years, many machine learning approaches have been used to generate embeddings, such as neural networks [19]. It has been demonstrated that a Transformer-based model [73] is a class of neural network that performs a variety of NLP tasks quite well [76]. To solve many issues, the architecture's design is crucial. Transformers are composed by two main blocks: encoders and decoders. A Transformer-based encoder (like BERT [22]) is able to produce embeddings from input sequences, such as text. Embeddings are vector representations of data points, such as words, phrases, in a continuous numerical space. These vector representations allow algorithms to work with textual more effectively, as they capture meaningful relationships and similarities between data points. By converting complex data into manageable numerical vectors, embeddings enable tasks like text classification, recommendation systems, and sentiment analysis. In our thesis, we use phrase-BERT [75] in order to create embeddings, since it is more adapted for creating embeddings of short phrases and keywords.

In order to group keywords similar to each other, words are considered as nodes within a network/graph and the similarity relationships between them are examined. There are several techniques for clustering text data that have been converted into embeddings, including **supervised clustering** and **unsupervised clustering**.

Finley et al. [25] state: *"Supervised clustering is the problem of training a clustering algorithm to produce desirable clusterings: given sets of items and complete clusterings over these sets, we learn how to cluster future sets of items."*. While supervised clustering uses labeled data [25] for the clustering process to improve the clustering findings accuracy, unsupervised clustering aims to group similar items without any prior knowledge or labels. Two popular unsupervised clustering techniques are HDBSCAN [51] and K-means [39]. In many cases, it can be easier to learn from previous and existing correlations between terms and group them using labeled data.

Text data can also benefit from community detection algorithms such as the Louvain method [11] and the Kruskal algorithm [45]. Kruskal's algorithm is primarily used for finding the minimum spanning tree of a graph, which can be seen as a form of community detection, as Haponchyk et al. use in their studies [33]. The Louvain method, on the other hand, is a community detection algorithm used to find communities within large networks. A variety of clustering metrics [2] can be utilized to assess the quality of the conclusions, such as F1 Score, ARI, NMI and Fowkles score.

1.2. Problem

A lot of sources can produce data and this has led to find effective methods to extract valuable information, that can be helpful for a lot of businesses. One of these methods is clustering, which plays a significant role by grouping similar objects and enabling the categorization of data. Moreover, the labels associated to the data are not always available since labelling require humans [65] and it could be a time consuming tasks.

The project is conducted in collaboration with Gavagai AB, a Swedish company focused on NLP field. The goal is to gather relevant insights from a vast amount of user keywords, especially we want to find out which the best model approach is (among the ones that will be presented in Sec. 2.5) that can be used to recommend clusters.

In the word clustering field, different unsupervised community detection algorithms can be used to identify relevant word groups [79]. When used for keyword clustering, they might not always produce the best results, leading to the creation of low-accurate clusters. To solve these challenges, one possible solution could be the use of supervised clustering training of a neural network and after it the use of community detection techniques. We use different clustering evaluation metrics which are analyzed in Sec. 2.4.

The research questions that we address in this thesis are the following:

- **RQ1**: *How do supervised clustering techniques compare to unsupervised clustering techniques in terms of clustering keywords, using evaluation metrics, such as NMI and ARI?*
- **RQ2**: *What are the differences in clustering performance metrics, such as NMI and ARI, when used to evaluate clusters generated by applying K-Means and HDBSCAN to keyword embeddings?*
- **RQ3**: *What are the key differences in the clustering performance metrics - NMI and ARI - when these metrics are utilized to assess the outcomes of applying the Louvain, Kruskal, and Greedy Modularity Maximization algorithms to keyword clustering?*

1.3. Purpose

This thesis investigates and explores the efficacy of incorporating supervised clustering techniques into the process of keyword clustering. By combining and incorporating additional data in the form of labels, the thesis aims to address the limitations of traditional unsupervised clustering methods and check whether supervised clustering may enhance the accuracy of the cluster's results.

The main objectives of this thesis are assessing the performances of different community detection algorithms in clustering keywords and investigating the impact of integrating the use of supervised clustering on the accuracy of keyword clusters.

The thesis aims to contribute to the advancement of keyword clustering techniques and the research outcomes will provide helpful insights for businesses seeking to extract meaningful information from large amounts of data, for instance improving recommendation systems. This study's results will also offer new directions for research in data analysis and clustering approaches by highlighting the possibility of combining supervised clustering with community detection algorithms.

1.4. Goal

The thesis seeks to investigate how community detection techniques may be applied to cluster keywords, represented as phrase-BERT embeddings [75], which captures the semantic meaning of words. The primary objectives are to assess the quality of the results using clustering metrics and to compare the effectiveness of two methods: supervised clustering and conventional unsupervised clustering algorithms.

These have been divided into the following sub-goals:

1. Clustering phrase-BERT embeddings using unsupervised community detection techniques, to group similar words together, enabling better categorization of textual data.

2. Training a model following a supervised learning method, using labeled data, to try to improve the accuracy of the cluster results.

3. Evaluating and comparing the different clustering approaches.

1.5. Benefits, Ethics and Sustainability

This thesis presents some benefits to those businesses (such as Gavagai AB) that deal with Machine Learning and NLP tasks. Thanks to the use of NLP techniques, it is easier to extract information from large amounts of text data. Keyword clustering helps businesses to categorize data, trying to extract powerful insights, and to improve their recommendation systems, leading to more accurate recommendations for the customers.

It is also important to handle data privacy: textual data often contains sensitive information such as personal details or confidential business data. Another problem could be dealing with different biased word associations, that can impact the clustering results, leading to unfair categorization [12].

1.6. Research Methodology

The research method involves the analysis of measurable data and it is also crucial to quantify and translate the results obtained by different models into numbers. We selected the empirical method because it enables us to test different models and draw conclusions based on how they perform, and to do so we need to utilize metrics to determine which approach is the most accurate, in a quantitative way.

In this thesis project, the objective is to compare the clustering metrics obtained with different experimental approaches. First of all, we train using a supervised clustering approach, and then using the community detection techniques we create clusters. Secondly, we cluster word phrase-BERT embeddings in an unsupervised approach. Finally, the research methodology ends with a comparison of the outputs and a discussion about the results.

1.7. Stakeholders

There could be different stakeholders that might be interested in the results and outcomes of this thesis, such as businesses, data scientists and researchers, which deal with NLP and word embeddings, but mainly the company which will use the results of this thesis project to find out a way to better recommend clusters to users. This research contributes to the improvement of keyword clustering techniques and remove up novel opportunities for investigation in supervised clustering algorithms.

1.8. Delimitations

One of the major limitations of this thesis is the choice of the right clustering metric to evaluate the results. Where there are many different methods for analysing the results obtained by the algorithms, it is fundamental selecting the appropriate metrics to provide relevant final explanations. In this thesis, since we have access to ground truth (labeled data), where the true cluster assignments are known, we use external evaluation metrics that compare the clustering results to the ground truth (as we discuss in Sec. 2.4).

Another limitation for future research regarding this topic is the use of a proper dataset, containing text and the labels associated to it. As we describe in Sec. 3.2, the company provided a helpful dataset containing all the information needed to conduct this research.

1.9. Outline

This thesis is organized as follows: in Chapter 2 we introduce the relevant theory and background regarding NLP, Supervised Clustering, Community Detection algorithms, the Loss functions and the related work. In Chapter 3, we present the details of the research methodology and then the data used for the experiments. Chapter 4 presents the implementation details. In Chapter 5 we present the results of the experiments, that are discussed in Chapter 6, where we propose future directions and work for the research. Appendix A contains additional results from the experiments.

2 | Theoretical Background

This chapter introduces the extended background of this thesis research, covering the theory needed for the project. Section 2.1 explains the theory about Natural Language Processing. Section 2.2 focuses on Transformers and Sentence Embeddings. Section 2.3 provides a general overview on clustering, with a focus on Supervised Clustering. Section 2.4 describes the evaluation metrics for assess the quality of the outcomes, obtained with different Community Detection Algorithms, described in the Section 2.5. Section 2.6 focuses on Contrastive Learning, used for running some experiments and Section 2.7 describes the loss functions of this research project. Finally Section 2.8 covers related works, regarding this topic.

2.1. Natural Language Processing

Natural Language Processing (NLP) [15] enables computers to understand and interpret human language in a meaningful and useful way. The input and output for a NLP system can be speech and/or written text.

The first research regarding NLP began in the 1950s as the intersection of artificial intelligence and linguistics [36, 55], and in the last twenty years it has been widely used for different tasks and purposes [16], such as sentimental analysis [72], topic modelling [41], information retrieving, language translation [36].

For performing these tasks, it is necessary to convert the raw text into a suitable representation that computer systems can understand. To convert unprocessed text into suitable representations for computer processing, there is a range of approaches, such as word embeddings [53] or sentence embeddings [22].

2.2. Transformer Architecture

The Transformer architecture was described for the first time by Vaswany et al. [73]. Transformer-based models are an appropriate neural network design for processing input sequences, being successful for many NLP projects. Recurrent Neural Networks (RNNs) [70] were replaced by transformer-based architectures for the resolution of tasks requiring the processing of input sequences. Transformers present many advantages over RNNs, such as the fact that Transformer models are unaffected by the vanishing gradient issue. Additionally, Transformers can handle entire input sequences, while RNNs process one part of the complete input sequence at a time.

Transformers are powerful models that are designed to process sequential data (sentences, documents) and can capture contextual information and relationships between words, using the Attention mechanism, originally described by Vaswani et al. [73]. This component captures relationships between words in a sequence, allowing models to focus on relevant parts of the input sequence. The Transformers are capable to convert text (such as words, sentences, and paragraphs) to a fixed-length dimensional dense vector.

Encoder and decoder parts compose the Transformer architecture. At a high level, the Transformer architecture utilizes a mechanism called self-attention to handle sequential data, such as text. Self-attention enables the model to take into account both local and global relationships between words by weighing the relative importance of various words in a sentence as each word is processed.

2.2.1. Sentence Embeddings

A sentence embedding is a numerical representation of a sentence that captures its meaning and context [64]. Unlike word embeddings, which represent individual words, sentence embeddings represent entire sentences. Word Embeddings technique focuses on individual words and provides effective representations for words in a continuous vector space, where words with the same meaning are represented similarly. Sentence Embedding models serve as a kind of interpreter between people and computers. These models, in particular, convert textual data (sentences) into a machine-understandable n-dimensional vector.

A global word embedding is learned by using conventional word embedding techniques. They begin by developing a general vocabulary using words that are specific to the documents while ignoring the meaning of words in various contexts. The words that appeared more frequently close to one another in the documents are then taught similar representations. The issue is that such word representations ignore the contextual meaning of the words. For instance, only one representation of the word "left" is learned when it appears in the sentence "*I left my phone on the left side of the table.*" However, because "*left*" has two different meanings in the sentence, the embedding space must also have two different representations for it. On the other hand, contextual embedding techniques take into account the order of all the words in the sentence to learn sequence-level semantics. Therefore, depending on their context, these techniques learn various representations for polysemous words, such as "*left*".

There are several ways to create sentence embeddings, such as BoW [66], TF-IDF [1] and Phrase-BERT [75]. Bidirectional Encoder Representations from Transformers (BERT) is a pre-trained transformer [73] network introduced in 2018 [22]. In contrast to earlier models that only took into account left-to-right or right-to-left context, BERT's bidirectional nature enables it to capture context from both directions (left and right) of a word in a sentence. The BERT model's Sentence-BERT (SBERT) [68] variant was created specifically to produce fixed-length sentence embeddings. While BERT and SBERT are both based on the Transformer architecture and have many similarities, SBERT is more suited to specific tasks and scenarios because it focuses on creating semantically meaningful sentence representations. For the purpose of this thesis, we use Phrase-BERT [75]. Instead of other sentence embedding models, Phrase-BERT performs well with short phrases and keywords, being helpful for our task.

Phrase-BERT

Sentence Transformers, such as Phrase-BERT [75], can be used to create word embeddings, mapping the words to a 768-dimensional dense vector space. Phrase-BERT is a simple but effective method to improve phrase embeddings from BERT using a contrastive learning approach (Sec. 2.6).

Wang et al. [75] show that Phrase-BERT embeddings can be easily integrated with a simple autoencoder to build a phrase-based neural topic model that interprets topics as mixtures of words and phrases by performing a nearest neighbor search in the embedding space. Phrase-BERT focuses on short text during pretraining, like phrases and words.

Using a contrastive learning approach, the authors have trained Phrase-BERT to place semantically similar phrases and words closer together and push apart different concepts, exploiting a triplet loss. Wang et al. [75] demonstrate that Phrase-BERT outperforms baselines (such as Sentence-BERT [68]) across a variety of phrase-level semantic tasks, while also showing increased lexical diversity between nearest neighbors in the vector space. Another advantage regarding the use of Phrase-BERT against BERT is the production of embeddings which do not raise any ethical concerns, regarding genders and racial biases [29].

Some examples of how the textual words are converted into arrays of numbers to be fed into the neural networks are provided in the table 2.1.

Term	**Sentence Embedding**
service	$[0.123, 0.456, 0.789, 0.321, ...]$
customer	$[0.987, 0.654, 0.321, 0.789, ...]$

Table 2.1: Example of word embeddings.

2.3. Clustering

Clustering is the task used to divide a dataset, composed by unlabeled data, such as nodes, and documents, into different groups or clusters, based on the similarity among the nodes. The aim of the clustering process is to have clusters such as nodes that are in the same group are more similar than nodes that are in different groups. It is an unsupervised learning method because it does not require the need for human intervention and it is used to process unclassified data objects into groups represented by patterns, to find meaningful structure.

We can define a cluster as a good cluster when the data points in the cluster are similar to each other (*small within-cluster variance*) and clusters are different from other clusters (*large between-cluster variance*) [7].

There are different techniques that can be used for clustering, such as K-Means [39], hierarchical clustering [18], and density-based clustering [24], each of them with strength and weaknesses.

2.3.1. K-Means

K-Means is a centroid-based clustering algorithm, computationally efficient and adapt to be used with large datasets. This algorithm divides the dataset into K clusters, where K is a hyperparameters defined in advance by the user. By assigning data points to the nearest centroid and updating the centroid positions until convergence, K-Means algorithm works iteratively. One of its drawbacks is that this algorithm needs to know in advance the number of K clusters and it is sensitive to the initial centroids selection.

One technique used to determine the optimal number of clusters is the **Elbow Method**. It involves plotting the number of clusters against the *within-cluster sum of squares* (WCSS), which measures the compactness of data points within each cluster. The *elbow point* in Fig. 2.1, where the rate of decrease in WCSS significantly slows down, suggests the

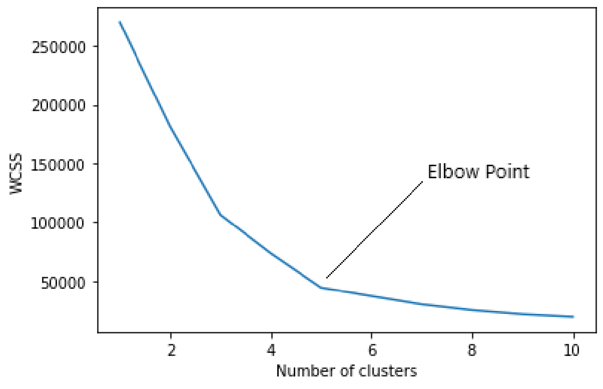


Figure 2.1: Elbow Method for selecting the optimal number of clusters.

appropriate number of clusters that balances minimizing WCSS and avoiding overfitting or underfitting.

2.3.2. Hierarchical clustering

Hierarchical clustering is based on the developing of the hierarchy of clusters in the form of a tree, called dendrogram. It is a representation of the clusters evolution (the merges of the clusters) during model training. This technique can be computationally expensive for large datasets and it starts by considering each data point as an unique cluster, then repeatedly merge clusters that are closest to each other until all data points are in a single cluster or a certain number of clusters are achieved.

2.3.3. Density-based clustering

Density-based clustering algorithms, such as DBSCAN (Density-Based Spatial Clustering of Applications with Noise), identify different clusters in the data, knowing that a cluster is a contiguous region of high point density, separated from other such clusters by other regions of low point density.

For this thesis project, as for the unsupervised approach, we use K-means with the Elbow method and HDBSCAN to cluster the keywords, without the use of human labels, and at the end we compare the results to see whether human labels are useful or not to create clusters.

HDBSCAN

HDBSCAN (Hierarchical Density-Based Spatial Clustering of Applications with Noise) [51] is a density-based clustering algorithm, created to overcomes some of DBSCAN's limitations [24], regarding parameters' choice (the tuning of Epsilon and Min Points involves a lot of trails) and the handling of different densities clusters. It extends the concepts of DBSCAN to provide a hierarchical clustering structure.

One of the advantages of HDBSCAN is that you do not need to specify in advance the number of clusters. This algorithm is composed by 5 steps:

1. Transform the space: the algorithm is used to find clusters in noisy data. It uses density estimation to distinguish denser regions ("land") from sparser regions ("sea"). It uses a core distance metric based on the distance to k-th nearest neighbor, to identify sea points an spread them apart. The mutual reachability distance metric is introduced to push sparse points away from each other, while keeping dense points' relative distances. It helps approximate the hierarchy of density distribution in single linkage clustering.

2. Build the minimum spanning tree: the mutual reachability metric is used to construct weighted graph representation of the data, in order to identify islands of dense data that can be the clusters that we are looking for. Adjusting a threshold value, edges with low weights can be removed, disconnecting the graph into connected components. Using Prim's algorithm, the minimum spanning tree of the graph is found and it represents the hierarchy of connected components, based on the mutual reachability distance.

3. Build the cluster hierarchy: the algorithm extracts the hierarchy of connected components from the minimum spanning tree found in the previous step.

4. Condense the cluster tree: the hierarchy is traversed and for each split the size of the new clusters is compared to an additional parameter: minimum cluster size. If the size is lower than the parameter, the points are marked as "points falling out of a cluster".

5. Extract the clusters considering the persistence of each cluster.

HDBSCAN has many advantages over other clustering algorithms. It has a few hyperparameters that can be tuned to achieve better results, such as the minimum cluster size, the minimum samples required to form a cluster, and the maximum distance between points in a cluster. It can discover clusters of varying shapes and sizes. This can be done through the use of a density-based strategy, which enables to find clusters based on the local density of the data points. It is able to find those points that do not belong to any clusters. This is useful in applications like outlier detection, where it's necessary to recognize data points that are different from the rest.

2.3.4. Supervised Clustering

Supervised Clustering (Sec. 1.1) is a type of clustering technique that uses labeled data to guide the clustering process. The goal is to create groups of similar data points using the known class labels or ground truth information. It is useful when there is prior knowledge about the structure of the data.

Supervised and unsupervised learning have a number of challenges, one of which is determining the quality of the results produced by clustering approaches.

2.4. Evaluation Metrics

Finding the right metrics to evaluate the results can be challenging, since several metrics can be used. Each of them has their own advantages and drawbacks. There are various types of metrics, depending on the type of task we are trying to accomplish and the type of data that we have. According to the presence or not of the ground truth labels, there are two different clustering evaluation metrics [21]:

- **Extrinsic Measures:** these metrics require ground truth labels to evaluate the performance of a clustering algorithm.
- **Intrinsic Measures:** these metrics do not require ground truth labels and evaluate the quality of clustering based on internal criteria, such as cluster compactness and separation.

2.4.1. Extrinsic Measures

Precision, Recall, and F1 score are common extrinsic measures [9, 56, 61], but there are also Rand Index, Mutual Information and Fowlkes-Mallows Score.
The aim of these evaluation metrics is to compare the ground truth, composed by the prior knowledge's labels from the dataset $P = \{P_1, P_2, \ldots, P_m\}$ to the label's results of the community detection algorithms, that are a potentially different partition of the data $C = \{C_1, C_2, \ldots, C_s\}$. The following terms are used to evaluate the performance of a clustering algorithm by comparing the predicted clusters with the ground truth clusters

[61].

Precision - Recall - F1 Score

Haponchyk et al. [34] use Precision and Recall (and the combination of both these measures: F1 Score) as two measures to define and compare the results of the analysis of the experiments. These both metrics have range [0, 1]. They compare the output clustering to the ground truth and to do so, they assign $\hat{c_j}$ to the most frequent gold class (cluster) and then compute the clustering's average **Precision** over the clustering as:

$$\text{Precision} \; = \frac{1}{N} \sum_{j=1}^{\hat{k}} \max_i |c_i \cap \hat{c}_j| \tag{2.1}$$

In Eq. 2.1, N is the number of points to be clustered and $\hat{k}$ the number of output clusters. This metric corresponds to the standard clustering purity defined Zhao et al. [81]. It is the percent of the total number of objects (data points) that were classified correctly. Instead, the **Recall** is defined as:

$$\text{Recall} \; = \frac{1}{N} \sum_{j=1}^{k} \max_i |\hat{c}_i \cap c_j| \, , \tag{2.2}$$

where k is the number of gold standard clusters.

Precision and Recall are both important measures of the performance of a clustering algorithm. More intuitively, one can say that precision measures how accurate the clustering algorithm is, while recall measures how complete the clustering algorithm is [2, 61]. Using Precision and Recall, it is possible to compute the clustering **F1 metric**, which is the harmonic mean of precision and recall.

$$\text{F1} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.3}$$

The F1 score is a good measure of both precision and recall, and it is often used as a single measure of the performance of a clustering algorithm [2].

As mentioned before, extrinsic measures require ground truth labels (for example thank to manual labeling by humans). Other metrics that can be used to evaluate the performances are Mutual Information, Rand Index and Fowkles-Mallows Scores [21].

Mutual Information

Mutual Information (MI) [50] is a function that measures the agreement of two assignments (in our case ground truth class and clustering algorithm), ignoring permutations. This measure is built on the Shannon Entropy of information theory: the entropy of a cluster is a measure of the homogeneity of the data points inside the cluster and allows the measurement of the degree of disorder in the clustering results [61]. It quantifies uncertainty and entropy decreases as the uncertainty decreases.

For a discrete random variable X with possible outcomes $x_1, x_2, ..., x_n$ and corresponding probabilities $p_1, p_2, ..., p_n$, the entropy is defined as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log p(x_i) \tag{2.4}$$

where $p(x_i)$ is the probability of a data point being classified as c in cluster x.

If a cluster has low entropy, it means that the data points are similar to each other and belong to the same class (high degree of homogeneity). The entropy of a cluster reflects how the members of the k categories are distributed within each cluster.

A higher score of MI means higher similarity. This measure tells you how these two assignments agree to each other, i.e., how much information they share about each other. This measure is symmetric, meaning that the mutual information between the variables X and Y is equivalent to Y and X: $I(X;Y) = I(Y;X)$.The measure ranges between 0 and 1 and the mutual information between two variables it is defined as:

$$I(X;Y) = H(Y) - H(X|Y) \tag{2.5}$$

where $H(X|Y)$ is the entropy of class labels Y within each cluster C. Substituing Eq. 2.4 in Eq. 2.5, the measure can also be written as:

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) log \frac{p(x,y)}{p(x)p(y)} \tag{2.6}$$

In Fig. 2.2, we can see how the mutual information is represented.

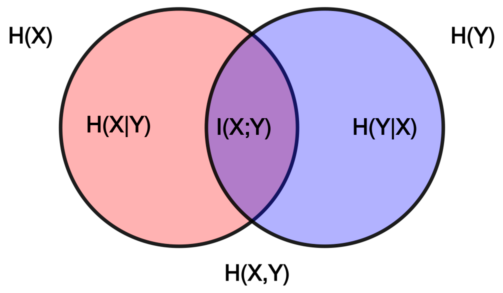


Figure 2.2: The intersection between the two circles that represent the individual entropies of the two variables is the mutual information I(X;Y).
Source: McDaid et al. [50]

Normalized Mutual Information

Normalized Mutual Information (NMI) [50] ranges between 0 (no mutual information) and 1 (perfect correlation) and it a symmetric function. A NMI value of 1 between two clusters means perfectly similar clustering, whereas a value of 0 means perfectly different clustering [47]. NMI is the mutual information divided by the average of the cluster entropies. The NMI value between two clusterings C and C' is measured as:

$$NMI(C, C') = \frac{2 \cdot I(C; C')}{H(C) + H(C')} \tag{2.7}$$

The Eq. 2.7 is derived by normalizing the mutual information by the sum of the entropies of the individual clusterings. One advantage of NMI is that it is easy to measure and compare the value between various different clusters having various numbers of nodes because it has been normalized [6]. Moreover, a permutation of the cluster label values will have no effect on the score value since this measure is independent of the labels' absolute values.

Rand Index

Rand Index (RI) [67] measures the similarity between two partitions, ignoring permutations, to evaluate the performance of a clustering algorithm. It is used to compare the actual class labels with the predicted cluster labels. Similar clusterings have a high Rand Index, which is a value between 0 and 1. The value 0 indicates that there is no agreement between the two data clusterings on any pair of points, and the value 1 means that they are precisely the same. It is defined as:

$$RI = \frac{a+b}{\binom{n}{2}} \tag{2.8}$$

In Eq. 2.8, a is the number of data points pairs that are assigned to the same cluster in both the predicted and true clustering, and b is the number of data points pairs assigned to different clusters in both the predicted and true clustering. n is the total number of data points. The binomial coefficient represents the number of unique pairs that can be formed by a set of n items and it can be represented by $n(n-1)/2$. One of the drawbacks of using RI is that it does not consider the possibility of agreement occurring by chance [54]. Two clusterings might have a high RI even if they have little in common and they are the result of random chance. For instance, if the cluster assignment was random, there might be different cases of "true negatives". We would like to have random label assignments to have scores close to 0, and to do so, we require adjusting for chance, using the following evaluation metric.

Adjusted Rand Index

Regardless of the number of clusters and samples [21], the Adjusted Rand Index (ARI) [37] guarantees a value near 0 for random labeling and an exact 1 value for identical clusterings. It ranges from -1 to 1, and -1 indicates that the two clusterings are completely different. ARI is still a measure of the similarity between two data clusterings, and it alters RI to take into account the likelihood that some agreement between two clusterings could happen by chance. This measure may have negative values for severely discordant clusterings because of the chance adjustment. This occurs when the agreement between the two clusterings is lower than what would be predicted if the cluster assignments were generated at random [13].

$$ARI = \frac{RI - E[RI]}{\max(RI) - E[RI]} \tag{2.9}$$

In Eq. 2.9, $E[RI]$ is the expected value of the Rand Index under the assumption of independence and $max(RI)$ represents the maximum possible value of the Rand Index.

If the results have an ARI close to 0, then it means that the result will be almost as good if you randomly permute all labels. A low ARI indicates a poor result.

Fowlkes-Mallows Scores

The Fowlkes-Mallows index (FMI) [28] is an evaluation metric and it is defined as the geometric mean between precision and recall:

$$\mathrm{FMI} = \frac{\mathrm{TP}}{\sqrt{(\mathrm{TP}+\mathrm{FP})(\mathrm{TP}+\mathrm{FN})}} \tag{2.10}$$

where:

- **True positives (TP):** the number of pairs of data points that are correctly assigned to the same cluster in both clusterings.
- **False positives (FP):** the number of pairs of data points that are incorrectly assigned to the same cluster in both clusterings.
- **True negatives (TN):** the number of pairs of data points that are correctly identified as belonging to different clusters.
- **False negatives (FN):** the number of pairs of data points that are incorrectly assigned to different clusters in both clusterings.

The range of FMI is between 0 and 1, where a score of 1 indicates a perfect match between the predicted and true cluster labels. The Fowlkes-Mallows score is based on the intersection and union of the two clusterings and it is a more robust measure than the F1 score, as it is not as sensitive to outliers.

Summary Evaluation Metrics

In Table 2.2, we summarize the metrics used for the experiments and for responding the research question, defined in Section 1.2.

Metric	**Description**
F1 Score	It is the harmonic mean of Precision and Recall [2].
ARI	It accounts for chance agreement and is suitable for comparing clusterings with varying sizes [37].
NMI	It normalizes the mutual information which quantifies the shared information between two clustering [50].
Fowlkes	It is a metric that calculates the geometric mean between precision and recall [28] .

Table 2.2: Summary of Clustering Evaluation Metrics used in this project.

2.5. Community Detection Algorithms

This section investigates the use of community detection algorithms in keyword clustering. The term *community* refers to a set of nodes in a network where nodes inside the community have more internal connections than external connections [27]. These algorithms are used to find and identify groups of nodes similar among them and they have different applications across various field, such as social network analysis, recommendation systems [82]. As we can see in Fig. 2.3, in this graph we have three different communities, represented by grey areas.

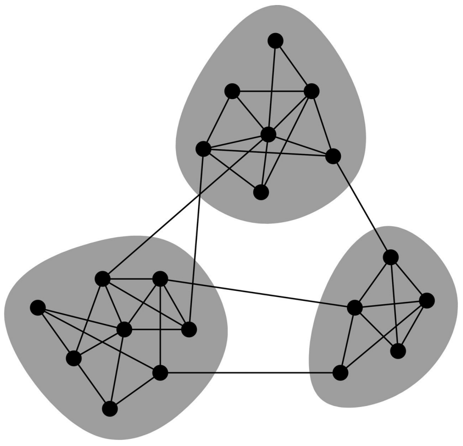


Figure 2.3: Communities in a graph.
Source: Newman et al. [59]

The graph networks can be represented in different ways: weighted and unweighted. In unweighted graphs, the edges between nodes indicate whether those nodes are connected or not, without considering any strength of the connections. On the contrary, weighted graphs are characterized by values on the edges between nodes, representing the similarity of the connections. To better represent the nodes n and their connections inside a graph G, we can use *adjacency matrices* [71]. An adjacency matrix A is a square matrix where a_{ij} indicates whether vertices i and j are adjacent (connected) in a graph.

For unweighted graphs a_{ij} is either 1 (indicating a connection) or 0 (no connection).

$$A \in \mathbb{R}^{n \times n}, a_{i,j} \in \{0, 1\}; i, j \in 1, ..., n$$

In the case of weighted graphs a_{ij} takes on numerical values representing the weight of the connection between nodes i and j.

$$A \in \mathbb{R}^{n \times n}, a_{i,j} \in \mathbb{R}; i, j \in 1, ..., n$$

n represents the number of nodes in a graph. When a graph is symmetric it is also defined as undirected: the presence of an edge between node A and node B implies the existence of an edge between node B and node A. The number of combinations N_c in a symmetric graph is the total amount of non-symmetric pairs of nodes n in the graph G.

$$N_c = \frac{n(n-1)}{2} \tag{2.11}$$

In the following sections, we discuss in detail about different algorithms that can be used for detecting clusters inside graphs.

2.5.1. Louvain

The Louvain method [11] is a widely used algorithm for detecting communities in networks. This is a heuristic method based on modularity optimization that run in time $O(n \cdot \log n)$, where n is the number of nodes in the network [46]. The *modularity* is a measure that is used to evaluate the quality of a partition in a graph and it a scalar value that ranges between -1 and 1. *"Modularity is a measure of the structure of a graph, measuring the density of connections within a module or community"* [59]. High modularity score means that the graph has many connections within a community, but only few connections that point outwards to other communities. In the weighted networks, the modularity is defined as [58]:

$$Q = \frac{1}{2m} \sum_{i,j} \left[\left(A_{ij} - \frac{k_i k_j}{2m} \right) \delta(c_i, c_j) \right] \tag{2.12}$$

Here A_{ij} represents the weight of an edge between the two nodes i and j; k_i and k_j represent the degrees of the nodes; c_i and c_j represent which is the community of nodes i and j, respectively; m represents the sum of all edge weights in the graph, and $\delta(\cdot,\cdot)$ is the delta function, which corresponds to 1 when c_i is equal to c_j and 0 otherwise.

This algorithm is divided in two phases that are repeated iteratively and its aim is to find high modularity partitions. In the first step, it assigns every node of a weighted network of n nodes to N different communities, to have the same number of communities and nodes. Afterwards, for each node, it moves each node to all of its neighbor communities, trying to find the maximum positive modularity. If there is no positive gain, the node remains in its original community. This process is applied repeatedly and sequentially for all nodes until there is no improvement. The final output of this algorithm is a hierarchical community structure.

2.5.2. Greedy Modularity maximization

Clauset et al. [17] have presented a hierarchical agglomeration algorithm for detecting communities in a graph, using a greedy modularity optimization. It proves faster than competing algorithms [17] with a complexity of $\mathcal{O}(md \cdot \log n)$, where n is the number of vertices in a network, m number of edges and d is the depth of the dendrogram that describes the community structure. This algorithm was created to be time efficient for large dataset, since the authors have developed it using a big Amazon dataset. The aim is to find the community partition with the largest modularity. Greedy modularity maximization begins with each node in its own community (hierarchical) and repeatedly merges the pair of communities that lead to the largest modularity until no further increase in modularity is possible.

2.5.3. Kruskal

Kruskal algorithm [45] is a greedy algorithm used to find the Minimum Spanning Tree (MST) of a weighted graph G. This algorithm can be also used for clustering, as Haponchyk et al. do in their studies [33]. In order to find clusters, Kruskal is used as an inference step to find a Maximum Spanning Tree h on G. This algorithm is based on the principle of choosing the available edge with the lowest/highest (according to the the chosen spanning tree) weight value that does not create a cycle. It finds an optimum solution at every stage of the iteration of the algorithm, instead of focusing on a global optimum, forming a minimum spanning tree with the selected edges. In a graph with E edges and V vertices, the time complexity is $\mathcal{O}(E \log E)$ or $\mathcal{O}(V \log V)$.

The Kruskal algorithm starts from edges with the lowest weight values, keeps adding the edges until the goal is reached. First of all, sort all the edges in a non-decreasing order of their weights. After it, take and add to the spanning tree the edge with the lowest weight, only if that edge does not create any cycles. Keep adding edges until all the vertices are reached, to obtain a minimum spanning tree at the end.

By computing a Maximum Spanning Tree h instead of a Minimum Spanning Tree, the algorithm finds those pairs of nodes with a high weight value on the edge. The higher the weight, the more similar the nodes and they will be placed together in the same cluster.

2.6. Contrastive Learning

Contrastive Learning [48] is a deep learning technique, based on the comparison between pairs of instances, given as input. It is used to learn representations of data in such a way that similar data points (positive pairs) have similar representations in embedding space, than the dissimilar data points (negative pairs). Samples are compared against each other, and those samples that belong to the same distributions are pushed toward each other in the embedding space, while the ones that have different distributions are pulled against each other. Two instances belonging to the same class lie close to each other in the embedding space are considered a positive example denoted d^+, and those belonging to different classes lie at a greater distance from each other are considered a negative, denoted d^-. Thus, a contrastive learning model tries to minimize the distance d^+ and maximize the distance d^-. So the goal is to learn how to discriminate data points that belong together from those that do not. Two word embeddings belonging to the same cluster should stay close to each other in the embedding space d^+ and those belonging to different clusters should stay at a greater distance from each other d^-. The contrastive learning model tries to minimize the distance d^+ and maximize the distance d^-.

Becker et al. [10] showed for the first time how to learn in a contrastive way using pairs of data as input: the similarity of positive pairs should be higher than the similarity of negative pairs.

Supervised Contrastive Learning (SCL) [42] is a variant of contrastive learning that incorporates supervised information into the learning process. SCL is a training framework that combines supervised learning with contrastive learning principles. By encouraging similar samples to be closer together in the learned feature space while pushing dissimilar samples apart, it aims to improve the representation learning process.

The machine learning models used for contrastive learning learn what makes the data points similar and dissimilar, using loss functions. The final goal for each contrastive loss function is to minimize the distance between samples.

2.7. Loss Functions

A loss function in a neural network is a function that compares the target y and predicted output values $\hat{y}$ (see 2.13). It measures the model's quality during training and offers a mathematical framework for updating the model's weights. The goal of the training process is to reduce this function in order to get the best possible model configuration.

$$loss = \mathcal{L}(\hat{y}, y) \tag{2.13}$$

According to the loss value, the model can be updated to get the best results. This section describes the two loss functions used in the thesis project: Binary Cross Entropy and Margin Ranking Loss.

2.7.1. Binary Cross Entropy

Binary Cross Entropy (BCE) is a loss function in machine learning, particularly used in binary classification tasks. It measures the dissimilarity between predicted probabilities and true binary class labels. The formula of the Binary Cross Entropy loss $\mathcal{L}_{BCE}$ is defined as:

$$\mathcal{L}_{BCE}(\hat{y}, y) = -\left(y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right) \tag{2.14}$$

In Eq. 2.14, y is the true binary label (0 or 1) of the sample and $\hat{y}$ is the predicted probability of the positive class (between 0 and 1). It has several advantages: it is a smooth and continuous function such that it can be optimized using gradient-based methods, and it is also convex, which means that it has a unique global minimum. By training a model using BCE, the output is composed the similarity (represented in percentage) between two nodes: two similar nodes should have a greater similarity value than two dissimilar nodes.

2.7.2. Contrastive Loss

Chopra et al. [14] created a similarity metric to keep similar points together and push dissimilar point, called Contrastive Loss. This metric-learning loss was used for the first time for dimensionality reduction [31]. The authors compare the pair of embeddings with Euclidean Distance: the smaller the distance, the closer the representations; the larger the distance between two dissimilar points, the more negative the loss becomes.

Margin Ranking Loss

The final goal for each contrastive loss function is to minimize the distance between samples. Margin Ranking Loss compares the distance between *positive pairs* (similar instances of the same cluster) and *negative pairs* (instances belonging to different clusters) in the feature space.

It measures the loss given two inputs $x1$, $x2$, and the target label y. The two inputs are the feature representations of two instances and the label contains either +1 or -1, used to indicate whether the instances are similar or dissimilar. If $y = +1$, the first input $x1$ should be ranked higher because it has a larger value than the second input $x2$, and vice-versa for $y = -1$. By adjusting y value, it is possible to bring similar nodes closer to each other and push away dissimilar nodes in a graph [34].
The Margin Ranking Loss function $\mathcal{L}_{MRL}$ for each pair of samples is defined as follows:

$$\mathcal{L}_{MRL}(x_1, x_2, y) = \max(0, -y \cdot (x_1 - x_2) + \text{margin}) \tag{2.15}$$

In Eq. 2.15, the *margin* is a hyperparameter that defines the minimum desired distance between positive and negative pairs.

2.8. Related Work

Haponchyk et al. [34] state *"Supervised clustering has been shown particularly effective for the NLP task"*. It has been shown that supervised techniques tend to be effective in grouping similar keywords together, which can be useful in improving search engine optimization, recommendation systems and content categorization. The use of supervised clustering techniques for keywords is a powerful tool for improving the accuracy of keyword grouping and it is expected to become even more prevalent in the coming years, with the increasing availability of data and the growing need for improved search and recommendation systems.

One possible approach to supervised clustering, analyzed in [33, 34] is using Neural Supervised Clustering (NSC) models, based on Latent Structural Support Vector Machines (LSSVM) and Latent Structural Perceptron (LSP) and on a latent representation of clusters using graph structures. In their studies, Lin et al. and Zhang et al. [78, 80] focused on intent clustering, which is about discovering new intents (expected questions) using limited knowledge over intent data, taking advantage of labeled data. Moreover, the authors design neural networks based on latent structured prediction loss and Transformer models to approach supervised clustering. Lin et al. and Zhang et al. test their methods on the task of automatically recreating categories of intents from publicly available question intent corpora and they found that their methods achieve state-of-the-art results. Their methods rely on pre-trained language models such as BERT (Bidirectional Encoder Representations from Transformers), which can be computationally expensive and domain-specific. In this degree project, we rely on using a pre-trained language model (Phrase-BERT, for its ability to properly embed both phrases and words) focused on clustering keywords.

Haponchik et al. [34] present a supervised clustering method based on hierarchical agglomerative clustering (HAC) with pairwise similarity scores derived from word embeddings. They find that their method outperforms baseline methods such as k-means clustering or HAC with cosine similarity on both datasets in terms of cluster purity and normalized mutual information (NMI). However, their method requires manual annotation of questions into intents for training data, which can be costly and time-consuming. The same authors also proposed a model for optimizing a structural clustering loss with neural networks, with the goal of training a model with a scoring function such that the correct clustering is scored higher than incorrect clusterings. Their approach is based on mapping the elements into nodes of a fully-connected undirected graph G for an input x, with nodes representing elements x_i of the input x and edges representing all pairwise links between them $(x_i, x_j)^2$. Any spanning forest h on G translates directly into a clustering y: the nodes in each linked component (spanning tree) of h are meant to be members of the same cluster. To find and detect the clusters, Haponchik et al. [34] used the Kruskal algorithm to find the maximum spanning forest. They have trained a basic simple feed-forward neural network, with ReLU activation functions. The pairwise encoder is trained to score positive edges higher than negative edges, which correspond to the pairs of questions that the authors have analyzed. However, none of the aforementioned papers tackling supervised clustering leverage community detection. Moreover, no one has discussed about supervised learning applied to keywords yet.

3 | Methodology

This chapter describes the strategies used to answer the research questions in Sec. 1.2.

In order to fully answer the research questions, we use a comparative approach to measure different models. At the end, we analyze and evaluate the results. To make the explanation and analysis of my experiments clearer in this chapter, I included several self-explanatory illustrations.

3.1. Research methods

This degree project follows a quantitative approach in order to answer the research question [38]. To do so, we employ the experimental research method, quantitatively evaluating the performance of different models and different hyperparameters. The evaluation will be done using the same data set, that will be discussed in Sec. 3.2, keeping any other experimental variable constant, for instance the training and testing split. The quality of the output clusters obtained by applying different algorithms is evaluated using the clustering metrics described in Sec. 2.4. The details of the research methodologies are shown in Fig. 3.1. In few words, we perform different experiments on the text embeddings. One of our goals is to see whether clustering word embeddings is better following a supervised learning technique or not. Based on the collected quantitative results, we draw our conclusions.

The output from the supervised learning approach is composed by adjacency matrices (see Sec. 2.5), which represent the graphs. From them, we can extract the clusters with community detection algorithm which will be evaluated with the evaluation metrics. In Fig. 3.2, we can see how the comparison and evaluation work. After that, we compare the results of the comparison to state the conclusions. The output from the unsupervised algorithms are clusters.

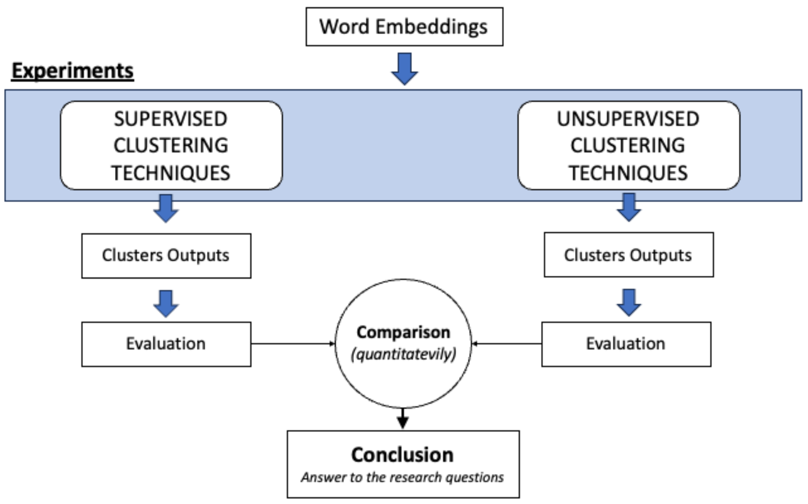

Word Embeddings
Experiments
SUPERVISED CLUSTERING TECHNIQUES
UNSUPERVISED CLUSTERING TECHNIQUES
Clusters Outputs
Clusters Outputs
Evaluation
Comparison (quantitatevily)
Evaluation
Conclusion
Answer to the research questions

Figure 3.1: The high-level overview of the research methodology.

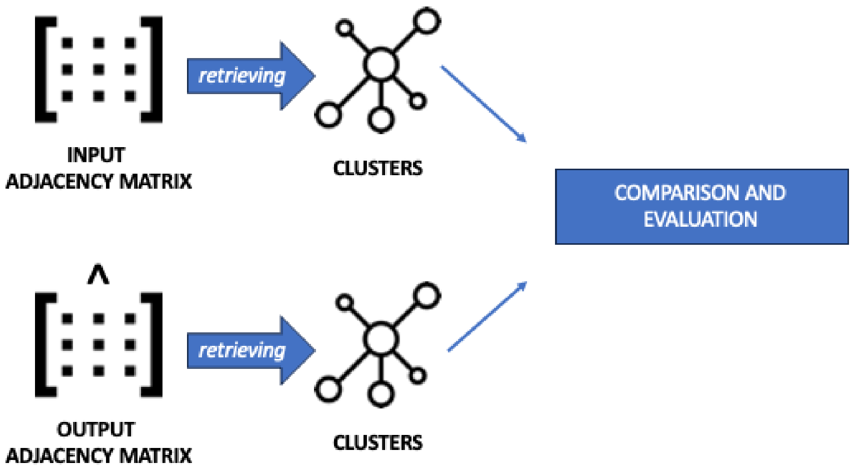


Figure 3.2: General view on the cluster evaluation.

3.2. Data Collection

To execute the experiments and facilitate the project, the company furnished me with a dataset comprising nearly fifteen million labeled keywords. It is composed by four different attributes described below:

A dataset example of the tuples can be found in table 3.2.

The average length of the text is one word, so it means that the dataset is mainly composed by keywords. In the dataset, there are 46 unique language values, such as English, French, Swedish, Italian and so on. For the purpose of this thesis, we focus only on the English

Attribute	Description
MD5	Indicates the keyword's label. If two keywords have the same MD5 value, it means that they belong to the same cluster.
Term	The keyword itself, the text data that will be converted into a word embedding.
UserId	Indicates the user who belongs to the text data.
Language	Specifies the language of the keyword.

Table 3.1: Description of the dataset's attributes.

MD5	Term	UserId	Language
`a0e8f907`	price	12345	EN
`b76cd2fe`	tjänst	67890	SV
`a0e8f907`	cost	12345	EN

Table 3.2: Example of the dataset.

terms, to be more consistent with the results, but as we will discuss in Sec. 6.2.1, regarding the future works of the project, it is possible to analyze another language to see whether the results are similar or not.

By selecting only the English terms, at the end for our experiments we have 465 unique users and 2 millions tuples. After having performed some data analysis, we notice how the dataset is mainly composed by business keywords, and that is why this degree project is suitable for those companies that deal with these terms.

To proceed with the experiments, we need to create another attribute which contains the embeddings values (Sec. 2.2) of the keywords. To do so, we apply phrase-BERT, explained in Chapter 4. Afterwards, these values are fed into the deep learning models and/or unsupervised algorithms that we analyze in the following sections.

3.3. Results Analysis

We need an adequate data analysis approach to assess the data after the results have been collected. The experiments to answer the research questions use quantitative research. We compare the evaluation metrics on the clusters obtained by the algorithms (Fig. 3.2): we obtain the clusters from the adjacency matrices which represent the graphs. To do so, as we analyze in Sec. 3.5, we need to run several experiments with different hyperparameters.

3.4. Model Design

In order to achieve the main purpose of this research, we need to define a pipeline, as we can see in Fig. 3.3.

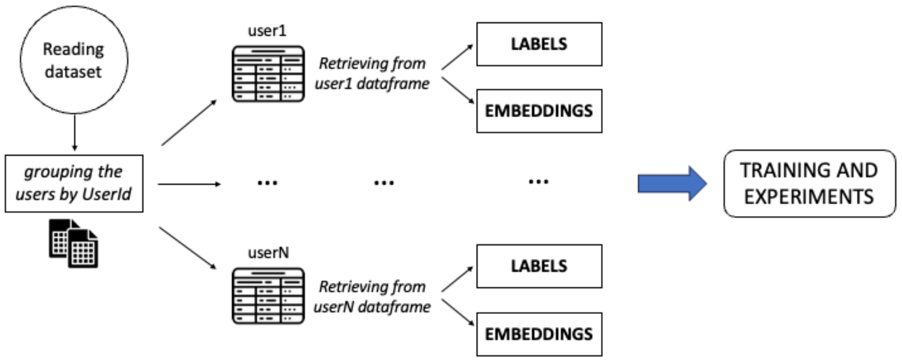

Reading dataset
grouping the users by UserId
user1
Retrieving from user1 dataframe
LABELS
EMBEDDINGS
...
...
...
userN
Retrieving from userN dataframe
LABELS
EMBEDDINGS
TRAINING AND EXPERIMENTS

Figure 3.3: General view of the project.

This pipeline is in common among the two experiments: first, we read the dataset and we group the N users to have N different dataset, containing all the M words belonging to the users. We can associate each training instance (an user dataset) to a graph. In this way, our training is not biased on only a dataset, but we train and obtain results from many datasets. After that, we retrieve from each dataset two values which will be use for the training:

- **Labels**: this 1-D tensor contains M labels associated to the keywords of that user, represented in a numerical way. For instance, if two keywords have the same label, they will have the same integer which means that belong to the same cluster.
- **Embeddings**: this 2-D tensor contains M word embeddings with length 768, each embedding corresponds to a word.

In Table 3.3, we can see a possible configuration of these two values which are used and combined in different ways according to the experiment that we want to run.

Labels:	`[0, 1, 1, 2]`
Embeddings:	`[[5.6, ...], [1.2, ...], [0.3, ...]]`

Table 3.3: Example of the user data.

For instance, this data extracted from an user states that the user has 4 keywords and there are 3 clusters (as we can see in the Labels value).

3.4.1. Supervised Learning

In Supervised Learning (Sec. 2.3.4) we need the labels associated to each word embedding. In Fig. 3.4, it is easier to visualize how supervised learning works in this thesis project. So, to run the experiments, we both need Labels and Embeddings, as previously described.

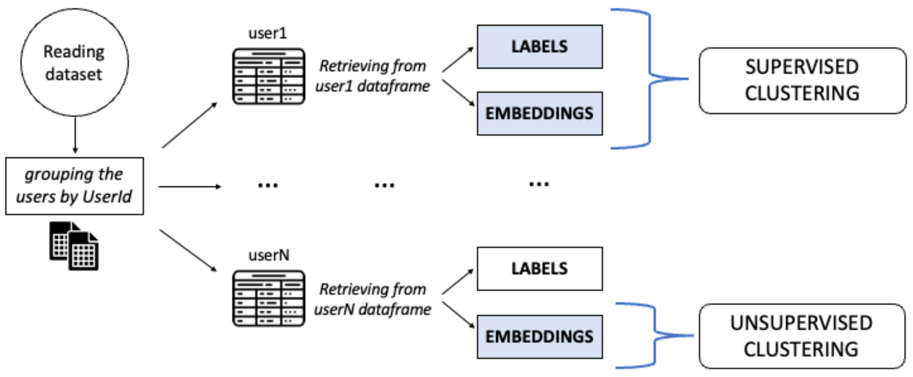


Figure 3.4: Differences between supervised learning and unsupervised learning.

3.4.2. Unsupervised Learning

In Unsupervised Learning the only thing required for the experiments is the word embeddings value, as we can notice in Fig. 3.4. Since it is unsupervised, we cannot use the labels that are provided with the embeddings.

In the following section, we provide more details about the experiments.

3.5. Experimental Design

This section describes the detailed settings of the two experiments in order to evaluate the results of the two different approaches.

The output of both the experiments are adjacency matrices which represent the graphs. Each graph is composed by nodes which represent the word embeddings. From the graph, it is possible to extract the clusters, using the clustering techniques explained in Sec. 2.5. Both the experiments are used for answering to **RQ1** (see Sec. 1.2).

Fig. 3.5 shows how the experiments are structured.

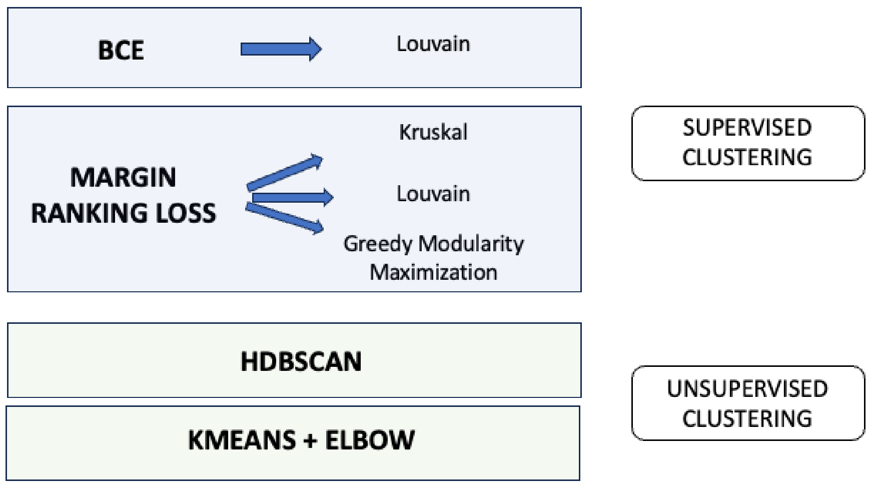


Figure 3.5: Supervised learning and unsupervised learning.

3.5.1. Experiment 1: Supervised Clustering

This experiment is based on training using supervised clustering and then applying community detection techniques to cluster the output (adjacency matrices). We need this experiment to answer to **RQ3** (see Sec. 1.2).

First of all, we need to find a way to compute the values on the edges e between the nodes in the graphs, with different neural network models.

Wang et al. [74] state deep learning models can be used with word embeddings. For this experiment and to gather many results, we use different Neural Network models (with many hyperparameters and layer configurations). Each user with his own data (labels and embeddings) is a training instance, so each step of the experiment is conducted for the N users available in the dataset. The experiments' steps are represented in Fig. 3.6 and they are the following:

1. We gather labels and embeddings belonging to the user x.
2. We transform labels and embeddings (explained below) to be given to a Neural Network model (the description of the Neural Network layers is provided in Sec. 4).
3. We train the model for several epochs, using one of the losses described in Sec. 2.7.

4. We collect the output.

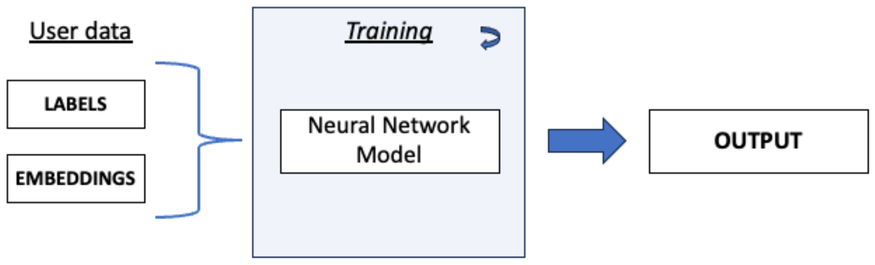


Figure 3.6: The supervised experimental approach.

For a training instance, the input of the neural network is composed by a tensor containing pairs of keywords (converted into word embeddings). So, in input the model receives a 3-D tensor **I**, where the first one describes the number of pairs (P), the second one states that the elements are pairs (2) and the third one is the word embedding dimensions (768).

$$\mathbf{I} \in \mathbb{R}^{P \times 2 \times 768}$$

We pass pairs of keyword/nodes, i.e., edges $e = (x_i, x_j)$ and we obtain from the model a 1-D tensor **O**, where the dimension correspond to the number of pairs P.

$$\mathbf{O} \in \mathbb{R}^{P}$$

The values in the output can be evaluated as either the *weight* (such as Haponchyk et al. [33] used for their experiments) or the *percentage similarity* between two nodes. From these output values, we reconstruct the adjacency matrices that represent the graphs containing the nodes. In Fig. 3.7, we see the input and output configuration for this experiment: the former is composed by pairs of word embeddings and the latter is composed by numerical values.

Since we are training models, our aim is to reduce the loss. According to the chosen loss function (Margin Ranking Loss or BCE - see Sec. 2.7) and the model's configuration, we use the provided labels associated to the keywords in different ways. According to the activation function at the end of the neural network, we obtain different range of values in output.

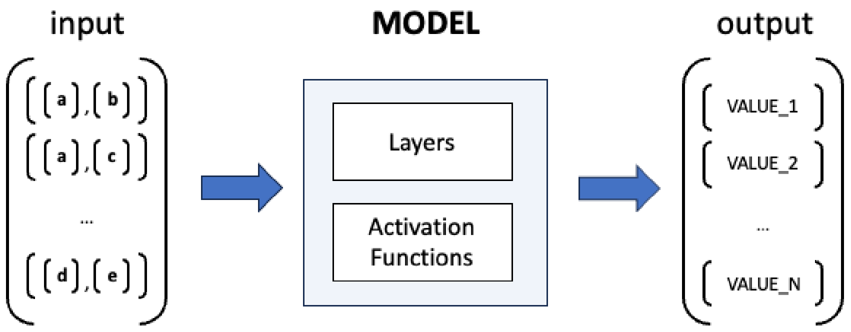


Figure 3.7: Input and output configuration.

ReLU and Margin Ranking Loss

With a *ReLU* layer at the end of the neural network, we obtain positive values greater or equal than 0, representing the **weight on the edge** between two nodes. The higher the weight, the more important the connection is. $\mathcal{L}_{MRL}$ (see Eq. 2.15) needs three values: x_1, x_2, y, where in our experiments x_1, x_2 are model's outputs, obtained from the model (which has a ReLu activation function at the end) that receives in input one tensor after the other one. The first input tensor i_1 contains all the pairs of text that have the same label, while the second one i_2 contains all the pairs of text with different labels. Feeding i_1 and i_2 to the model, we obtain two different output tensors x_1 and x_2 which contain numerical values.

At the end, we obtain a model that is trained on putting similar words in the same area and different words far from each other.

Sigmoid and Binary Cross Entropy

The goal is to have nodes within clusters that have greater similarity values than the node connections between clusters. To do so, we need to have in output an activation function that ranges between 0 and 1, such as *Sigmoid*. The input given to the model is composed by the N_c pairs of keywords (see Eq. 2.11), since the graphs are undirected.

BCE (Sec. 2.7.1) measures the dissimilarity between the predicted probabilities and the true labels. In our experiments, the two variables of $\mathcal{L}_{BCE}$ (see Eq. 2.14) coincide with the model's output (a tensor containing the similarity between the N_c pair of nodes) and a tensor containing only 1 or 0. If a pair of nodes is in the same cluster the value corresponds to 1, otherwise 0.

Community Detection Algorithms

After having trained different models with many configurations and loss functions, we obtain and store the best model b, according to the minimum validation loss obtained. Afterwards, this model is tested on the testing set: we pass to b the pairs of words of each testing user, without the labels. After it, we analyze the output obtained by the model. To do so, we transform the input and the output (with length N_c) into adjacency matrices. Since the graphs are weighted and undirected, we only need N_c values that corresponds to the upper triangle and consequently the lower triangle. Along the diagonal we can set the values equal to 1, since every node is surely similar or stay in the same cluster as itself.

In Fig. 3.2, we can see that to compute the evaluation we need to retrieve the clusters and to do so, we can run some community detection algorithms (Sec. 2.5) on the adjacency matrices which represent the graphs.

The main idea about applying **Kruskal's algorithm** is to find the maximum spanning tree to cluster the nodes. By doing this, we can find the nodes having a stronger edge connectivities, i.e., they are more similar to each other. So this algorithm can be applied only on the output obtained by the models with the Margin Ranking Loss, which use a ReLU activation function as final layer.

Regarding **Louvain** and **Greedy Modularity maximization**, we only need to apply these algorithms on the graphs, and they will extract different clusters in different computing time. For the purpose of this thesis, we mainly focus on the model which uses the Margin Ranking Loss (as Haponchyk et al. have done in their experiments [34]). Therefore, for the BCE model we use only Louvain as community detection algorithm and we apply all of three algorithms to the Margin Ranking Loss model.

3.5.2. Experiment 2: Unsupervised Clustering

These experiments are used to answer to **RQ2** (see Sec. 1.2). As we can see in Fig. 3.8, we only need the word embeddings.

In these experiments, we pass all the word embeddings of each user at a time to the algorithms (K-means or HDBSCAN). By doing this, the algorithms create and find different clusters for each user, as for the Experiment 1.

These experiments should be faster than the first ones since the algorithms only need the embeddings and they directly create the clusters. At the end, the clusters will be

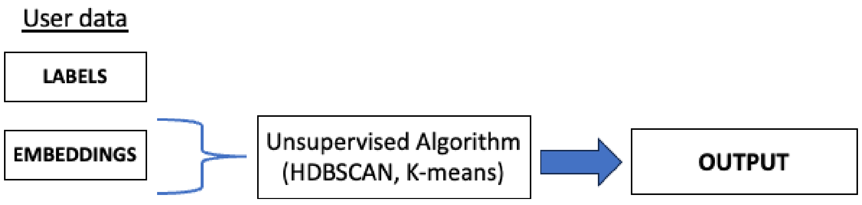


Figure 3.8: The unsupervised experimental approach.

evaluated using the same cluster evaluation technique, used in the Experiment 1 as well. We want to replicate the cluster evaluation in Fig. 3.2 and to do so, we need the input adjacency matrices. In this case, the output clusters are obtained by the algorithms, while the input adjacency matrices are obtained looking at the dataset, focusing on the "MD5" attribute. The adjacency matrix of a user u will be a square matrix with length n which correspond to the number of words of user u, where rows and columns correspond to the nodes of the graph, in our project the nodes are the word embeddings. If two words have the same label, in the adjacency matrix there will be a 1, indicating a connection between the two nodes. If the graph is weighted, there could be other positive values to indicate the weight.

4 | Implementation

In the previous chapter, we analyzed the methodologies and the theory regarding my degree project. Now, let's analyze it deeper. This chapter explains the technical details about the experiments, specifying the models' architectures, the hyperparameters and how to replicate our work.

4.1. Hardware and Software

The project is written using Python language, which comprises many useful libraries. For this thesis project, we used different libraries from the most common ones, used in data analysis and machine learning (such as `pandas` [52], `numpy` [35], `torch` [62]) to the more specific ones:

- `sklearn` [63]: from this library we use two different libraries: `metrics` is used for the evaluation metrics (Sec. 4.5), such as:
  - `adjusted_rand_score`
  - `normalized_mutual_info_score`
  - `fowlkes_mallows_score`

  The submodule `clusters` is used for performing KMeans, for running the Unsupervised Experiment.
- `igraph` [20]: powerful library used for creating, manipulating, and analyzing graphs. In our case, it is used to create a graph given an adjacency matrix.
- `sentence_transformers` [68]: used for transforming the text data into word embeddings. The details about it are explained in Sec. 4.2.
- `networkx` [32]: provides many functions to perform community detection (other details are provided in Sec. 4.3.4).
- `hdbscan` [51]: necessary to compute the Unsupervised Experiment, regarding HDB-

SCAN.

For running the experiments, the company provided me the dataset and two GPUs: NVIDIA RTX A5000 (24GB) and NVIDIA GeForce RTX 3080 Ti (12GB). These GPUs are particularly useful since they allow to run both the experiments simultaneously.

In the following sections, we describe all the technical details which are utilized for providing a better explanation, regarding the experiments. First of all, after having read the dataset we need to have the word embeddings.

4.2. Sentence Embeddings

Since many terms are repeated in the dataset and to perform this operation in an efficient and fast way, we have created a dictionary containing all the unique terms in the dataset. After it, we have converted them into the sentence embeddings storing the value in the dictionary. In this way, we have a dictionary with the keys that are the terms and the values are the sentence embeddings. For doing so, we need to use the library `sentence_transformer` [68], in particular for this project Phrase-BERT is used. Phrase-BERT creates meaningful phrase embeddings by fine-tuning BERT and as Wang et al. say [75], it outperforms strong baseline models.

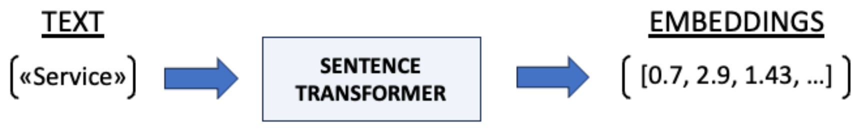


Figure 4.1: General view about how a Sentence Transformer works: it takes some text and it converts it into an array of values.

In the Phrase-BERT case, the architecture transforms the word into an array with length 768, meaning that each word or phrase has 768 features (also known as dimensions).

After having stored all the word embeddings in a data structure, since we train a model for the Supervised Experiment, we need to split the dataset in different subsets (Sec. 4.3.1).

4.3. Supervised Learning Experiments

4.3.1. Dataset Splitting

We divide our dataset into three different subsets with different length: training, testing and validation. Briefly, the **training set** is used to train the machine learning model, which learns by adjusting its parameters to minimize the prediction errors. The **validation set** is a subset of the data used during the training to fine-tune the model's hyperparameters. It helps in detecting overfitting (when a model performs well on the training set but poorly on new data). The **testing set** is a separate portion of the data that is used to evaluate the performance of a trained model. It is not used during the training process to ensure an unbiased evaluation. The model makes predictions on the testing set, and its evaluation metrics, are calculated in order to drawn some analysis. In our supervised experiments, the proportion of training/testing/validation is 70/20/10. First of all, we need to retrieve the number of unique users in the dataset: 487. After that, we compute the above percentages to create the three subset. To do so, we randomly take from the dataset the 70% of 487 for creating the training set and so on (see Table 4.1).

One issue observed is about those training instances (users u) with a large number of words. As we perform and store word combinations, the total number of items we must store grows, yield scalability issues due to memory limitations. To solve it, we must use random sampling [60], which involves selecting words at random by those users u.

	Training Set	Testing Set	Validation Set
Number of unique Users	341	97	49

Table 4.1: Dataset splitting.

By doing so, we have three different subsets with no overlapping users among them. It is better to remember that each user with his words is a training instance and it represents a graph. So in our case it is similar to have 341 different small dataset used to train our model. We prevent bias by ensuring that the three dataset splits do not share any users between them.

Another operation to do is the one used to remove those training users that have either only one word or less than two clusters, since we cannot cluster anything with those ones. After this kind of preprocessing operation, for our supervised experiment we have 313 users (see Table 4.2).

As for the unsupervised experiment, we do not need to split the dataset into subsets, but

	Training Set	Testing Set	Validation Set
Number of unique Users	**313**	97	49

Table 4.2: Dataset splitting after data cleaning.

we only have to group by the users in order to create several instances. They will be given to one of the two unsupervised algorhtms.

4.3.2. Model Training

The steps for training a model are the following ones:

1. Defining the optimizer: method used to adjust the parameters of a model during the training process. Its scope is to minimize the loss function by iteratively updating the model's parameters based on the gradients of the loss with respect to those parameters. In our project, we use Adaptive Moment Estimation (Adam). It allows to use a fixed amount of memory regardless of the size of the training set, and in our case the memory management is an important aspect to consider.
2. Defining the learning rate (`lr`), which is a hyperparameter that determines the step size at each iteration during the optimization process. Since it is a hyperparameter it needs to be tune with different experiments.
3. Training with several epochs using an early stopping condition to prevent overfitting. It involves monitoring the performance of the model on a validation set during training and stopping the training process when the performance on the validation set starts to degrade.

Hyperparameter	**Value**
Learning Rate	0.0001
Epochs	500
Patience	15
Optimizer	Adam
Loss Functions	$\mathcal{L}_{BCE}$, $\mathcal{L}_{MRL}$

Table 4.3: Hyperparameters for model training.

According to the chosen loss function, there are different model configurations with hyperparameters which will be analyzed below. Some standard hyperparameters are defined in Table 4.3.

The layers used for defining the deep learning models are described in 4.4. These layers are combined together to obtain different models which will be discussed below.

Layer Description	**Formula**
A **Linear Layer** operates by carrying out a linear transformation on the input data. This involves computing the product of the input vector $\mathbf{x}$ and a set of adjustable weights (represented by the matrix $\mathbf{W}$), and then adding a bias vector $\mathbf{b}$ [23].	$\mathbf{y} = \mathbf{W}\mathbf{x} + \mathbf{b}$
A **ReLU** activation function introduces non-linearity to the model by setting the negative values of the input vector $\mathbf{x}$ to zero and leaving positive values unchanged. It helps the model learn complex patterns and improves its ability to approximate non-linear relationships [3].	$\text{ReLU}(\mathbf{x}) = \max(0, \mathbf{x})$
A **Sigmoid** activation function squashes the input values $\mathbf{x}$ between 0 and 1, providing a probabilistic interpretation. It is often used in binary classification problems to produce a probability value that represents the likelihood of a particular class [57].	$\sigma(\mathbf{x}) = \frac{1}{1+e^{-\mathbf{x}}}$

Table 4.4: Description of layers and functions in the model.

Model used with Margin Ranking Loss

The Margin Ranking Loss $\mathcal{L}_{MRL}$ is defined in Eq. 2.15. For computing this training with $\mathcal{L}_{MRL}$, three values are needed: x_1, x_2, y. In order to perform this training, we follow these steps:

- We identify the indices of those terms that have the same labels and we store the embeddings with those indices in a tensor called *sl*. We create another tensor called *dl* that corresponds to those embeddings with different labels. Now we have two tensors of different lengths which contain different things: one contains all the pairs of embeddings with the same label and the other one all the pairs of embeddings that have different labels among each other.
- Since the tensors have different lengths, we determine the size that both of the tensors should have to perform downsampling on the longer tensor. So, now we have two tensors with the same length.
- We pass the *sl* and *dl* through the model to obtain the outputs x_1 and x_2, respectively.

- We create a target tensor called y and we fill it with "1", to give more importance to the first tensor in $\mathcal{L}_{MRL}$.
- We calculate the loss with the Eq. 2.15.

In order to have as many results as possible to run better analysis, we used different models' configuration (see Table 4.5).

Hyperparameter	Value
Hidden Size	512, 1024
Layers	2, 3
Margin γ	1, 10

Table 4.5: Hyperparameters for training with Margin Ranking Loss.

Using these hyperparameters, we obtained two different models, represented in Fig. 4.2.

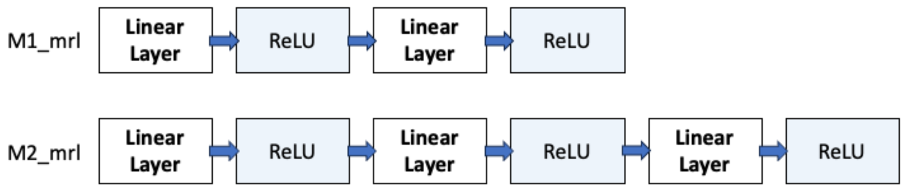


Figure 4.2: Neural Network Architectures for Margin Ranking Loss.

Every model (with its initial hidden size and with different margin - see Eq. 2.15) produced different results which will be showed in Chapter 5.

Model used with BCE

The BCE Loss function $\mathcal{L}_{BCE}$ is defined in 2.14. For computing the training with BCE, two values are required: $\hat{y}$ and y, as discussed in Sec. 2.7.1. In our case, $\hat{y}$ is the output obtained from the model and y represents the ground truth of the label values and it is computed looking at the tensor which contains all the labels for the training instance.

The steps for this training are represented in Listing 4.1:

```
# dataset: collection of data samples used containing labels and embeddings associated to a user
# device:  hardware on which computations are performed (CPU or GPU)

```

```
for labels, embeddings in dataset:
    labels, embeddings = labels.to(device), embeddings.to(device)

    combinations = generate_combinations(embeddings.shape)
    adj = calculate_adjacency_matrix(labels)
    input = select_input_pairs(embeddings, combinations)
    true = select_true_labels(adj, combinations)
    output = model(input)
    loss = loss_bce(output, true)

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

Listing 4.1: A PyTorch pseudocode about training using BCE

In order to have as many results as possible to run a better analysis, we used different models' configurations (see Table 4.6).

Hyperparameter	Value
Hidden Size	64, 512, 1024
Layers	2, 3
Activation Functions among the layers	Sigmoid, ReLU

Table 4.6: Hyperparameters for training with BCE.

Using these hyperparameters, we obtained four different models, represented in Fig. 4.3.

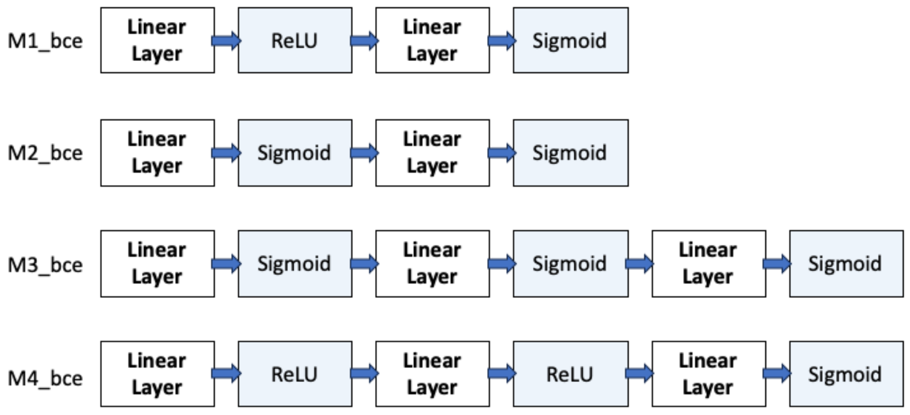


Figure 4.3: Neural Network Architectures for BCE.

Every model (with its initial hidden size) produced different results which will be showed in Chapter 5.

4.3.3. Creation of Output Adjacency Matrix

After having stored the best model (the one with the lowest validation loss) among the epochs, we test it on the testing set. We pass to the model the terms' combinations of the testing instances and we store the output. After it, we recreate the adjacency matrices from the outputs. Each output has dimension N_c (Eq. 2.11) and it corresponds to the upper triangle of the adjacency matrix. Since the graph is undirected, the upper triangle corresponds to the lower triangle.

4.3.4. Community Detection Algorithms

After having retrieved the weighted graphs G from the adjacency matrices using the library `networkX`, we apply the three community detection techniques, described in Sec. 2.5 on the graphs G:

- **Louvain**: we use the function `community.best_partition` on G [11].
- **Greedy Modularity maximization**: for this algorithm we use the function `greedy_modularity_communities` on G that uses Clauset-Newman-Moore greedy modularity maximization [17].
- **Kruskal**: in order to perform this algorithm, we need to find the maximum spanning tree on G using the function `maximum_spanning_tree`. In this way we obtain those clusters that contain nodes with the largest weights.

All of these three algorithms are utilized with the model used with Margin Ranking Loss. As for the model used with BCE Loss, we use only Louvain since Kruskal is preferable when there are weights and not probabilities. After having computed these techniques, we use an algorithm that sort and assign the labels of the clusters to have a final tensor similar to the input one. So instead of having a tensor such as $[1, 1, 3, 3, 3, 0, 2]$, we have a tensor that does not depend on specific label IDs, transforming into the tensor $[0, 0, 1, 1, 1, 2, 3]$. In this way, we have an output tensor similar to the input one which will be compared (Sec. 4.5).

4.4. Unsupervised Learning Experiments

After having as many graphs as the number of users, we need to compute the unsupervised techniques on the instances which will produce clusters as output.

4.4.1. HDBSCAN

As we said in Sec. 2.3.3, the advantage of this algorithm is that we do not need to specify in advance the number of clusters in a graph. So, for each user we only have to extract his word embeddings (i.e. nodes in the graph) and computing the algorithm with the library (see Sec. 4.1).

In order to find the best configuration of hyperparameters (see Table 4.7 for this technique, the grid search method is leveraged.

Hyperparameter	**Value**
`min_cluster_size` α	2, 10, 200
`min_samples` β	2, 10, 200

Table 4.7: Hyperparameters for HDBSCAN algorithm.

4.4.2. K-Means using Elbow Method

The main drawback about this algorithm is the need to specify the number of clusters in advance. But as we analyzed in Sec. 2.3, we can use the Elbow Method to find the optimal number of clusters K. By doing so, we can perform the algorithm knowing the exact number of K for each user. At the end we obtain the L clusters extracted from each graph. In order to run this algorithm we use the python function `KMeans` from `sklearn.cluster`. We pass to the function the word embeddings of each user/instance (which corresponds to a graph) and we use the elbow method to determine K. The elbow method plots the inertia for different numbers of clusters, and the optimal number of clusters is the point where the curve starts to bend sharply.

4.5. Evaluation

From each algorithm and from each graph, we obtain a tensor $\hat{y}$. It contains the predicted labels of the clusters. To make it clearer, let's do an example. We have a graph with six nodes and three clusters such as the first cluster contains the nodes 1,2,3; the second cluster contains the nodes 4,5 and the third cluster contains node 6. The tensor $\hat{y}$ has size 6 (like the number of nodes): `[0,0,0,1,1,2]`. This means that the first three elements have the same cluster because the label value is the same, and so on. It will be compared with a tensor obtained by looking at the MD5 attribute values. By checking the labels' values we create a tensor y that has the same size as the number of nodes/words in an instance. Each evaluation metric function takes in input $\hat{y}$ and y and produces an output value.

We store all the values of the metrics for every instance available in the testing set. Afterward, we can compute the average or the weighted average to have the final results regarding each algorithm.

4.5.1. Average

In order to have the average of each metric, we compute the average among all the values obtained. So for instance to have the final value of the metric ARI, we need to compute it for each instance and at the end determine the average of all the stored ARI values.

By doing so, we are not considering that the instances have different numbers of clusters so we also need to consider the dimension (i.e. number of words) in the instances.

4.5.2. Weighted Average

For quantifying the weighted average, we examine the number of terms, giving more importance to those users with more terms. This approach emphasizes the impact of the number of nodes in the graph on the overall evaluation score. We count the number of terms in each graph and we give to that user a weight. The higher the number of nodes the higher the value. In this way, we differentiate users with few terms and users with a huge number of terms.

5 | Results and Discussion

This chapter contains only the *best results* collected by the experiments described in the previous chapters. In order to have a larger view of the outcomes, *all the results* are visualized in Appendix A.

Firstly, we analyze the outcomes coming from the supervised learning experiments and secondly, the unsupervised learning outcomes are analyzed. Each metric has two different values: the average value and the weighted value (as described in Sec. 4.5). We also offer an analysis of another measure that indicates how long an algorithm takes and may be used to do further investigations. For evaluating the performances, we consider the following evaluation metrics: **F1 Score**, **NMI**, **ARI** and **Fowkles Score**, described in Sec. 2.4.

As discussed in Sec. 4.5.2, we give priority to those models with a higher NMI and ARI values, in order to define which the best model is.

5.1. Experiment 1: Supervised Learning

For the Supervised Learning experiment - described in Sec. 4.3 - six different models are used. The four models trained with **BCE Loss** (Sec. 4.3.2 - models architecture in Fig. 4.3) are defined with the following notation: M^1_{BCE}, M^2_{BCE}, M^3_{BCE}, M^4_{BCE}. The other two models trained with the **Margin Ranking Loss** (Sec. 4.3.2 - models architecture in Fig. 4.2) are defined with M^1_{MRL}, M^2_{MRL}.

Every model is trained with different hyperparameters (Sec. 4.3.2). In order to be more concise, only the results obtained by the best models are showed (for a complete view of the results, see Appendix A.

5.1.1. Comparison BCE - Margin Ranking Loss

In order to do a better comparison between these two approaches, we compare and illustrate in Table 5.1 the best results obtained by the best BCE Model configurations (Table

A.2) and the best Margin Ranking Loss Model (Table A.4). We highlight the higher values to make the comparison easier to visualize.

The models analyzed are M^2_{BCE} trained with `hidden_size = 1024` and using Louvain and M^2_{MRL} trained with `hidden_size = 1024` and $\gamma = 10$ and using Kruskal.

Weighted Average Metric	M^2_{BCE}	M^2_{MRL}
F1 Score	0.229	**0.447**
NMI	0.473	**0.771**
ARI	0.108	**0.296**
Fowlkes	0.195	**0.336**
Time (sec)	**8,054.13**	18,117.27

Table 5.1: Comparison between the best BCE and Margin Ranking Loss models.

5.1.2. Comparison Community Detection Algorithms

Table 5.2 displays the results and differences of the community detection algorithms, applied to M^2_{MRL}. These numerical outcomes are taken from Table A.4.

Algorithm	**Weighted Average of Evaluation Scores**				**Time (s)**
	F1 Score	NMI	ARI	Fowlkes Score	
Kruskal	**0.447**	**0.771**	**0.296**	**0.336**	18,117.27
Louvain	0.13	0.351	0.053	0.111	8,912.74
Greedy	0.098	0.212	0.023	0.094	**1,659.41**

Table 5.2: Evaluation scores and time for the community detection algorithms.

5.2. Experiment 2: Unsupervised Learning

For the Unsupervised Learning experiment (described in Sec. 4.4), there are two algorithms: HDBSCAN and K-Means with Elbow Method. In the following sections, we show the best outcomes obtained by the experiments.

5.2.1. Comparison HDBSCAN - K-Means

In order to do a better comparison between the unsupervised learning algorithms, we compare the evaluation metrics obtained by the best HDBSCAN configurations and K-Means. The best HDBSCAN configuration is picked from Table A.8: $\alpha = 200$ and $\beta = 200$.

The results are illustrated in Table 5.3. We highlight the higher values to make the comparison easier to visualize.

Weighted Average Metric	HDBSCAN	K-Means
F1 Score	0.444	**0.59**
NMI	**0.693**	0.423
ARI	0.112	**0.234**
Fowlkes Score	0.143	**0.442**
Time (sec)	**1,554.36**	6,128.37

Table 5.3: Comparison between HDBSCAN and K-Means.

5.3. Comparison Supervised and Unsupervised Learning

In the previous experiments, we have plotted the results obtained by different approaches. In order to run our final conclusions and discussions, Table 5.4 is showed. It comprises the two supervised (BCE and Margin Ranking Loss) and unsupervised learning approaches (HDBSCAN and K-Means).

Weighted Average Metric	Supervised Learning		Unsupervised Learning	
	BCE	Margin Ranking Loss	HDBSCAN	K-Means
F1 Score	0.229	0.447	0.444	**0.59**
NMI	0.473	**0.771**	0.693	0.423
ARI	0.108	**0.296**	0.112	0.234
Fowlkes score	0.195	0.336	0.143	**0.442**
Time (sec)	8,054.13	18,117.27	**1,554.36**	6,128.37

Table 5.4: Comparison of best metric values for supervised and unsupervised Learning

5.4. Discussion

One of the delimitations of this degree project (Sec. 1.8) is choosing the right evaluation metric to run the final analysis and to define the best model. For the purpose of this thesis project, since we have graphs of different dimensions, we focus on **NMI** (described in Sec. 2.4.1) and **ARI** (described in Sec. 2.4.1), considered with the **weighted average**. These measures are powerful when comparing clusterings of various sizes are needed and they are the appropriate evaluation measure to use for running conclusions of this thesis project since the graphs and clusters of our experiments have varying sizes. Since NMI is normalized we can measure and compare the value between different clusterings having different numbers of clusters. ARI gives us an idea of whether the clusterings are similar to what would be expected by random chance (if ARI is close to 0). We choose to prioritize the weighted average since the graphs analyzed in this degree project have different dimensions, so the number of nodes inside a graph is a metric that should be considered when running the analysis.

The experiments were conducted using different metrics to evaluate the models and analyze their results. After having plotted many outcomes, a proper final discussion about them is crucial. One of our final aim is to check which approach performs better to cluster keywords in terms of evaluation metrics against the others. In order to do, we have compared the best models gathered from the experiments in Appendix A. We mainly focus on the outcomes obtained by computing the weighted average (Sec. 4.5.2) of the evaluation metrics, to highlight the importance of the algorithms to consider the size of the graphs. It is important to note that there is no universally "best" metric among the ones that we have provided for running our analysis. The choice of the metric depends on the specific context and the nature of the data. Each metric captures different aspects of clustering quality and has its own strengths and limitations. We should also highlight the fact that these results come from one dataset, so it is possible that these approaches bring to different results on a different dataset.

Firstly, we compare and display in Table 5.1 the results of the best model trained using BCE and the best model trained using Margin Ranking Loss. For the purpose of our task, the model trained using Margin Ranking Loss with Kruskal performs better than the other Supervised Learning approach. M^2_{MRL} shows superiority in each evaluation metric, especially on the NMI value (0.771 against 0.473). The higher NMI indicates that the Margin Ranking Loss model preserves the mutual information between clusters better, contributing to a more accurate representation of the data's underlying structure. The main difference between these two approaches is the lower ARI value (0.108) for the BCE approach, stating that this is not the right approach for this kind of task. ARI measures the similarity of data points present in the clusters i.e., how similar the instances that are present in the cluster. So, this measure should be as high as possible else we can assume that the data points are randomly assigned in the clusters.

A comparison on the community detection algorithm is provided in Table 5.2. Kruskal is the one with the best results, in every evaluation metric, except for the computing time. Clauset et al. [17] with their Greedy Modularity Maximization algorithm, developed a fast community detection algorithm and in our experiments Greedy is faster than the other algorithms (1,659.41 seconds), but it provides the worst outcomes. In order to understand which unsupervised algorithm is better for clustering keywords, Table 5.3 is provided. K-Means is better among almost all the evaluation metrics, except for NMI. HDBSCAN has a higher weighted average NMI of 0.693, surpassing K-Means' weighted average NMI of 0.423. The higher ARI value of K-Means (0.234) suggests better clustering results, against 0.112 obtained by HDBSCAN.

Table 5.4 displays a summary of all the best algorithms used to figure out which is the best approach for clustering keywords. K-Means performs better in terms of F1 Score (0.59) and Fowlkes Score (0.442). On the other hand, Margin Ranking Loss with Kruskal shows superiority in ARI (0.296). A higher ARI indicates that the Margin Ranking Loss model provides a better adjustment for chance when measuring the agreement between clusters. Margin Ranking Loss has the highest NMI value (0.771), but there is no huge difference between the NMI value obtained with HDBSCAN (0.693). HDBSCAN has one of the worst ARI result (0.112) and having this evaluation metric value close to 0 (see Sec. 2.4.1) means that the nodes are assigned into the clusters as in a random way, as explained by Hubert et al. [37]. This analysis underscores the advantages of MRL and highlights the advantages and drawbacks of HDBSCAN and K-Means in unsupervised learning scenarios.

6 | Conclusions and Future Works

This chapter wraps up the thesis by providing some insights into the proposed strategy.

6.1. Conclusion

This study compares two approaches regarding clustering keywords, in a supervised way or in an unsupervised way. We have computed and analyzed many experiments to answer to the research questions (Sec. 1.2). In the context of **Supervised Learning**, where the availability of labeled data and higher accuracy are prioritized, the choice of community detection algorithm becomes crucial. In our experiments, the model trained using the **Margin Ranking Loss** and **Kruskal** emerged as the top-performing algorithm in every evaluation metric, achieving a NMI value of 0.771, against the 0.351 of Louvain and 0.212 of Greedy. It outperformed also regarding the ARI value (**0.296**) against 0.053 and 0.023. This indicates that Kruskal effectively provided accurate clustering results and it is the best community detection algorithm, associated with a supervised learning approach, to cluster keywords. On the other hand, even the unsupervised learning algorithm (K-Means with the Elbow Method) can be used as a viable alternative.

By looking at the results obtained in Appendix A, among these algorithms, HDBSCAN shows the lowest ARI values, meaning that the clusters obtained are not as valid as the clusters obtained by other algorithms. This means that K-Means (with an ARI value of 0.234) is more reliable if we want to use an unsupervised approach, instead of HDBSCAN which produces clusters almost similar to random clusterings.

According to Romano et al. [69], the main cause for the low ARI values is given by the fact that our ground truth clusterings associated to each user may have large different-sized clusters. The authors state that it is better to use ARI when the ground truth clusterings have large equal-sized clusters and to use Adjusted Mutual Information (AMI) when the ground truth clustering is unbalanced and there exist small clusters. The highest NMI obtained value (0.771) might be considered a low value since NMI tends to favor solutions with a high number of small clusters, such as Amelio et al. state [5].

HDBSCAN and BCE yielded the least favorable results in all the metric evaluations, with ARI values of 0.108 and 0.112, respectively. These low ARI values suggest that their outputs should be treated with caution, as they resemble "random" clusters, lacking significant structure or coherence. The low ARI values observed in these clustering approaches may be attributed to the complexity of the data or suboptimal algorithm selection. Addressing these factors through careful data preprocessing, and algorithm tuning could potentially improve the clustering quality. As a result, in cases when a small decrease in accuracy is acceptable, **K-Means** might be an appropriate solution since it produces relatively accurate clustering results. On the other hand, supervised learning with the Kruskal algorithm would be the recommended approach if getting the maximum accuracy is the major objective.

By looking at the results, some improvements can be made in the future in order to get better outcomes and to visualize some output cluster examples, obtained by the experiments. This research study can be used as a baseline to implement future algorithms which might lead to better outcomes results.

6.2. Limitations

This section covers some of the project's limitations. To begin with, selecting the appropriate assessment metric necessitated multiple investigations on the datasets and results. As we previously discussed, there are many extrinsic measures and each of them has its advantages and disadvantages.

Dealing with a large dataset also presented a substantial time barrier. Time-consuming tasks included applying the sentence transformer to the dataset and then training the models. Additionally, the wide variety of hyperparameters increased the time contribution because they needed to be optimized and fine-tuned, which took a lot of effort.

6.2.1. Future Work

Future study might look at other datasets and algorithms to confirm these results and give more thorough suggestions for selecting the best technique for clustering tasks in various settings. Although this work contributed significantly to the area, there are various improvements for future research that can build on the results and improve our understanding even further:

- **Using other evaluation metrics**: other metrics could be exploited such as Fair

Normalized Mutual Information (FNMI) [5] and Adjusted Mutual Information (AMI).

- **Exploring alternative Sentence Transformers**: while phrase-BERT was used as the sentence transformer in this study, other sentence transformers should be considered. This investigation might help determine whether alternative sentence transformers can increase clustering performance.
- **Investigating alternative community detection algorithms**: this study focused on the application of community detection algorithms on weighted graphs. However, further investigation into alternative community detection algorithms can be helpful. Other clustering techniques for the unsupervised approach can be applied, such as OPTICS.
- **Improving training**: since there are many different models' configurations, future works can deal about tuning other hyperparameters. It is also possible to make the supervised learning approach faster, adopting new algorithms. Moreover, instead of the single train-validation-test split, a cross validation technique can be explored.
- **Exploring other languages**: in this study, the focus was primarily on English words and their word embeddings. A potential avenue for future research is to extend the analysis to other languages.
- **Considering stemming for word preprocessing**: stemming is a technique used to reduce words to their base or root form and it has the potential to improve the clustering results by reducing noise within the clusters. Future research can explore the effectiveness of stemming techniques on keyword clustering tasks using word embeddings.
- **Investigating the use of label propagation**: it is a clustering machine learning method that uses semi-supervised learning. When working with data that includes examples with labels and without labels, it is helpful. Label propagation's basic idea is to spread labels from labeled data points to unlabeled ones, using the underlying structure or similarity of the data points in order to draw conclusions or organize the data.

We can expand our understanding and contribute to the growth of knowledge in this field by addressing the points just raised.

Bibliography

[1] TF–IDF. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning*, pages 986–987. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_832. URL https://doi.org/10.1007/978-0-387-30164-8_832.

[2] E. Achtert, S. Goldhofer, H.-P. Kriegel, E. Schubert, and A. Zimek. Evaluation of Clusterings – Metrics and Visual Support. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1285–1288, Arlington, VA, USA, Apr. 2012. IEEE. ISBN 978-0-7695-4747-3 978-1-4673-0042-1. doi: 10.1109/ICDE.2012.128. URL http://ieeexplore.ieee.org/document/6228189/.

[3] A. F. Agarap. Deep learning using rectified linear units (relu), 2019.

[4] B. Alabdullah, N. Beloff, and M. White. Rise of big data – issues and challenges. pages 1–6, 04 2018. doi: 10.1109/NCG.2018.8593166.

[5] A. Amelio and C. Pizzuti. Is normalized mutual information a fair measure for comparing community detection methods? In *2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM)*, pages 1584–1585, 2015. doi: 10.1145/2808797.2809344.

[6] A. Amelio and C. Pizzuti. Is Normalized Mutual Information a Fair Measure for Comparing Community Detection Methods? In *Proceedings of the 2015 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2015*, pages 1584–1585, Paris France, Aug. 2015. ACM. ISBN 978-1-4503-3854-7. doi: 10.1145/2808797.2809344. URL https://dl.acm.org/doi/10.1145/2808797.2809344.

[7] L. M. Aouad, N.-A. Le-Khac, and T. Kechadi. Variance-based Clustering Technique for Distributed Data Mining Applications, Mar. 2017. URL http://arxiv.org/abs/1703.09823. arXiv:1703.09823 [cs].

[8] D. S. Asudani, N. K. Nagwani, and P. Singh. Impact of word embedding models on text analytics in deep learning environment: a review. *Artificial Intelligence Review*,

56(9):10345–10425, Sept. 2023. ISSN 1573-7462. doi: 10.1007/s10462-023-10419-1. URL `https://doi.org/10.1007/s10462-023-10419-1`.

[9] A. Banerjee, C. Krumpelman, J. Ghosh, S. Basu, and R. J. Mooney. Model-based overlapping clustering. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 532–537, Chicago Illinois USA, Aug. 2005. ACM. ISBN 978-1-59593-135-1. doi: 10.1145/1081870.1081932. URL `https://dl.acm.org/doi/10.1145/1081870.1081932`.

[10] S. Becker and G. E. Hinton. Self-organizing neural network that discovers surfaces in random-dot stereograms. *Nature*, 355(6356):161–163, Jan. 1992. ISSN 0028-0836, 1476-4687. doi: 10.1038/355161a0. URL `https://www.nature.com/articles/355161a0`.

[11] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, oct 2008. doi: 10.1088/1742-5468/2008/10/P10008. URL `https://dx.doi.org/10.1088/1742-5468/2008/10/P10008`.

[12] T. Bolukbasi, K.-W. Chang, J. Zou, V. Saligrama, and A. Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, page 4356–4364, Red Hook, NY, USA, 2016. Curran Associates Inc. ISBN 9781510838819.

[13] J. E. Chacón and A. I. Rastrojo. Minimum adjusted Rand index for two clusterings of a given size. *Advances in Data Analysis and Classification*, 17(1):125–133, Mar. 2023. ISSN 1862-5347, 1862-5355. doi: 10.1007/s11634-022-00491-w. URL `https://link.springer.com/10.1007/s11634-022-00491-w`.

[14] S. Chopra, R. Hadsell, and Y. LeCun. Learning a Similarity Metric Discriminatively, with Application to Face Verification. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 539–546, San Diego, CA, USA, 2005. IEEE. ISBN 978-0-7695-2372-9. doi: 10.1109/CVPR.2005.202. URL `http://ieeexplore.ieee.org/document/1467314/`.

[15] K. R. Chowdhary. *Natural Language Processing*, pages 603–649. Springer India, New Delhi, 2020. ISBN 978-81-322-3972-7. doi: 10.1007/978-81-322-3972-7_19. URL `https://doi.org/10.1007/978-81-322-3972-7_19`.

[16] G. G. Chowdhury. Natural language processing. *Annual Review of Information Science and Technology*, 37(1):51–89, Jan. 2005. ISSN 00664200. doi: 10.1002/

aris.1440370103. URL https://onlinelibrary.wiley.com/doi/10.1002/aris.1440370103.

[17] A. Clauset, M. E. J. Newman, and C. Moore. Finding community structure in very large networks. *Phys. Rev. E*, 70:066111, Dec 2004. doi: 10.1103/PhysRevE.70.066111. URL https://link.aps.org/doi/10.1103/PhysRevE.70.066111.

[18] V. Cohen-Addad, V. Kanade, F. Mallmann-Trenn, and C. Mathieu. Hierarchical clustering: Objective functions and algorithms, 2017.

[19] R. Collobert and J. Weston. A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning.

[20] G. Csardi and T. Nepusz. The igraph software package for complex network research. *InterJournal*, Complex Systems:1695, 11 2005.

[21] S. A. Curiskis, B. Drake, T. R. Osborn, and P. J. Kennedy. An evaluation of document clustering and topic modelling in two online social networks: Twitter and reddit. *Information Processing Management*, 57(2):102034, 2020. ISSN 0306-4573. doi: https://doi.org/10.1016/j.ipm.2019.04.002. URL https://www.sciencedirect.com/science/article/pii/S0306457318307805.

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1423. URL https://aclanthology.org/N19-1423.

[23] A. D. Dongare, R. R. Kharde, and A. D. Kachare. Introduction to Artificial Neural Network. 2(1), 2012.

[24] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.

[25] T. Finley and T. Joachims. Supervised clustering with support vector machines. In *Proceedings of the 22nd international conference on Machine learning - ICML '05*, pages 217–224, Bonn, Germany, 2005. ACM Press. ISBN 978-1-59593-180-1. doi: 10.1145/1102351.1102379. URL http://portal.acm.org/citation.cfm?doid=1102351.1102379.

[26] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010.

ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2009.11.002. URL `https://www.sciencedirect.com/science/article/pii/S0370157309002841`.

[27] S. Fortunato. Community detection in graphs. *Physics Reports*, 486(3):75–174, 2010. ISSN 0370-1573. doi: https://doi.org/10.1016/j.physrep.2009.11.002. URL `https://www.sciencedirect.com/science/article/pii/S0370157309002841`.

[28] E. B. Fowlkes and C. L. Mallows. A method for comparing two hierarchical clusterings. *Journal of the American Statistical Association*, 78:553–569, 1983.

[29] D. Gala, M. O. Khursheed, H. Lerner, B. O'Connor, and M. Iyyer. Analyzing gender bias within narrative tropes. In *Proceedings of the Fourth Workshop on Natural Language Processing and Computational Social Science*, pages 212–217, Online, Nov. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.nlpcss-1.23. URL `https://aclanthology.org/2020.nlpcss-1.23`.

[30] D. Greene, P. Cunningham, and R. Mayer. Unsupervised Learning and Clustering. In M. Cord and P. Cunningham, editors, *Machine Learning Techniques for Multimedia: Case Studies on Organization and Retrieval*, Cognitive Technologies, pages 51–90. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-75171-7. doi: 10.1007/978-3-540-75171-7_3. URL `https://doi.org/10.1007/978-3-540-75171-7_3`.

[31] R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, volume 2, pages 1735–1742, New York, NY, USA, 2006. IEEE. ISBN 978-0-7695-2597-6. doi: 10.1109/CVPR.2006.100. URL `http://ieeexplore.ieee.org/document/1640964/`.

[32] A. Hagberg, P. Swart, and D. Chult. Exploring network structure, dynamics, and function using networkx. 01 2008.

[33] I. Haponchyk and A. Moschitti. Supervised Neural Clustering via Latent Structured Output Learning: Application to Question Intents. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3364–3374, Online, June 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.naacl-main.263. URL `https://aclanthology.org/2021.naacl-main.263`.

[34] I. Haponchyk, A. Uva, S. Yu, O. Uryupina, and A. Moschitti. Supervised Clustering of Questions into Intents for Dialog System Applications. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2310–2321,

Brussels, Belgium, 2018. Association for Computational Linguistics. doi: 10.18653/v1/D18-1254. URL `http://aclweb.org/anthology/D18-1254`.

[35] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585 (7825):357–362, Sept. 2020. ISSN 1476-4687. doi: 10.1038/s41586-020-2649-2. URL `https://www.nature.com/articles/s41586-020-2649-2`. Number: 7825 Publisher: Nature Publishing Group.

[36] J. Hirschberg and C. D. Manning. Advances in natural language processing. *ARTIFICIAL INTELLIGENCE*.

[37] L. Hubert and P. Arabie. Comparing partitions. *Journal of Classification*, 2(1): 193–218, Dec. 1985. ISSN 0176-4268, 1432-1343. doi: 10.1007/BF01908075. URL `http://link.springer.com/10.1007/BF01908075`.

[38] A. Håkansson. Portal of Research Methods and Methodologies for Research Projects and Degree Projects. *Computer Engineering*, 2013.

[39] X. Jin and J. Han. K-Means Clustering. In C. Sammut and G. I. Webb, editors, *Encyclopedia of Machine Learning*, pages 563–564. Springer US, Boston, MA, 2010. ISBN 978-0-387-30164-8. doi: 10.1007/978-0-387-30164-8_425. URL `https://doi.org/10.1007/978-0-387-30164-8_425`.

[40] Z. Kaddari, Y. Mellah, J. Berrich, M. G. Belkasmi, and T. Bouchentouf. Natural language processing: Challenges and future directions. 2020.

[41] P. Kherwa and P. Bansal. Topic modeling: A comprehensive review. *ICST Transactions on Scalable Information Systems*, 7:159623, 07 2018. doi: 10.4108/eai.13-7-2018.159623.

[42] P. Khosla, P. Teterwak, C. Wang, A. Sarna, Y. Tian, P. Isola, A. Maschinot, C. Liu, and D. Krishnan. Supervised contrastive learning. *CoRR*, abs/2004.11362, 2020. URL `https://arxiv.org/abs/2004.11362`.

[43] D. Khurana, A. Koli, K. Khatter, and S. Singh. Natural language processing: state of the art, current trends and challenges. *Multimedia Tools and Applications*, 82(3): 3713–3744, Jan. 2023. ISSN 1380-7501, 1573-7721. doi: 10.1007/s11042-022-13428-4. URL `https://link.springer.com/10.1007/s11042-022-13428-4`.

[44] J. Koirala. Understanding the Use of Cluster Analysis in Business, Mar. 2023. URL `https://papers.ssrn.com/abstract=4400674`.

[45] J. B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1):48–50, 1956. ISSN 0002-9939. doi: 10.2307/2033241. URL `https://www.jstor.org/stable/2033241`. Publisher: American Mathematical Society.

[46] A. Lancichinetti and S. Fortunato. Community detection algorithms: A comparative analysis. *Physical Review E*, 80(5):056117, Nov. 2009. doi: 10.1103/PhysRevE.80.056117. URL `https://link.aps.org/doi/10.1103/PhysRevE.80.056117`. Publisher: American Physical Society.

[47] A. Lancichinetti, S. Fortunato, and J. Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, Mar. 2009. ISSN 1367-2630. doi: 10.1088/1367-2630/11/3/033015. URL `https://dx.doi.org/10.1088/1367-2630/11/3/033015`.

[48] P. Le-Khac, G. Healy, and A. Smeaton. Contrastive representation learning: A framework and review. *IEEE Access*, 8:193907–193934, 01 2020. doi: 10.1109/ACCESS.2020.3031549.

[49] T. Madhulatha. An overview on clustering methods. *IOSR Journal of Engineering*, 2, 05 2012. doi: 10.9790/3021-0204719725.

[50] A. F. McDaid, D. Greene, and N. Hurley. Normalized Mutual Information to evaluate overlapping community finding algorithms, Aug. 2013. URL `http://arxiv.org/abs/1110.2515`. arXiv:1110.2515 [physics].

[51] L. McInnes, J. Healy, and S. Astels. hdbscan: Hierarchical density based clustering. *Journal of Open Source Software*, 2(11):205, Mar. 2017. ISSN 2475-9066. doi: 10.21105/joss.00205. URL `https://joss.theoj.org/papers/10.21105/joss.00205`.

[52] W. McKinney. pandas: a foundational python library for data analysis and statistics. 2011.

[53] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed Representations of Words and Phrases and their Compositionality.

[54] L. C. Morey and A. Agresti. The Measurement of Classification Agreement: An Adjustment to the Rand Statistic for Chance Agreement. *Educational and Psychological Measurement*, 44(1):33–37, 1984. ERIC Number: EJ297546.

[55] P. M. Nadkarni, L. Ohno-Machado, and W. W. Chapman. Natural language processing: an introduction. *Journal of the American Medical Informatics Association*, 18(5):544–551, Sept. 2011. ISSN 1067-5027, 1527-974X. doi: 10.1136/amiajnl-2011-000464. URL `https://academic.oup.com/jamia/article-lookup/doi/10.1136/amiajnl-2011-000464`.

[56] M. P. Naik, H. B. Prajapati, and V. K. Dabhi. A survey on semantic document clustering. In *2015 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT)*, pages 1–10, Mar. 2015. doi: 10.1109/ICECCT.2015.7226036.

[57] S. Narayan. The generalized sigmoid activation function: Competitive supervised learning. *Information Sciences*, 99(1):69–82, 1997. ISSN 0020-0255. doi: https://doi.org/10.1016/S0020-0255(96)00200-9. URL `https://www.sciencedirect.com/science/article/pii/S0020025596002009`.

[58] M. E. J. Newman. Analysis of weighted networks. *Physical review. E, Statistical, nonlinear, and soft matter physics*, 70 5 Pt 2:056131, 2004. URL `https://api.semanticscholar.org/CorpusID:1054844`.

[59] M. E. J. Newman. Modularity and community structure in networks. *Proceedings of the National Academy of Sciences*, 103(23):8577–8582, June 2006. doi: 10.1073/pnas.0601602103. URL `https://www.pnas.org/doi/10.1073/pnas.0601602103`. Publisher: Proceedings of the National Academy of Sciences.

[60] S. Noor, o. Tajik, and J. Golzar. Simple random sampling. 1:78–82, 12 2022. doi: 10.22034/ijels.2022.162982.

[61] J.-O. Palacio-Niño and F. Berzal. Evaluation Metrics for Unsupervised Learning Algorithms, May 2019. URL `http://arxiv.org/abs/1905.05667`. arXiv:1905.05667 [cs, stat].

[62] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

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

[64] C. S. Perone, R. Silveira, and T. S. Paula. Evaluation of sentence embeddings in downstream and linguistic probing tasks. *ArXiv*, abs/1806.06259, 2018. URL https://api.semanticscholar.org/CorpusID:49306018.

[65] B. Plank. The "problem" of human label variation: On ground truth in data, modeling and evaluation. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 10671–10682, Abu Dhabi, United Arab Emirates, Dec. 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.emnlp-main.731. URL https://aclanthology.org/2022.emnlp-main.731.

[66] W. Qader, M. M. Ameen, and B. Ahmed. An Overview of Bag of Words;Importance, Implementation, Applications, and Challenges. pages 200–204, June 2019. doi: 10.1109/IEC47844.2019.8950616.

[67] W. M. Rand. Objective Criteria for the Evaluation of Clustering Methods. *Journal of the American Statistical Association*, 66(336):846–850, Dec. 1971. ISSN 0162-1459. doi: 10.1080/01621459.1971.10482356. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1971.10482356. Publisher: Taylor & Francis _eprint: https://www.tandfonline.com/doi/pdf/10.1080/01621459.1971.10482356.

[68] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks, 2019.

[69] S. Romano, N. X. Vinh, J. Bailey, and K. Verspoor. Adjusting for Chance Clustering Comparison Measures.

[70] R. M. Schmidt. Recurrent neural networks (rnns): A gentle introduction and overview, 2019.

[71] H. Singh and R. Sharma. Role of Adjacency Matrix & Adjacency List in Graph Theory. *INTERNATIONAL JOURNAL OF COMPUTERS & TECHNOLOGY*, 3 (1):179–183, Aug. 2012. ISSN 2277-3061. doi: 10.24297/ijct.v3i1c.2775. URL https://www.cirworld.com/index.php/ijct/article/view/2775.

[72] M. T. H. K. Tusar and M. T. Islam. A comparative study of sentiment analysis using nlp and different machine learning techniques on us airline twitter data. pages 1–4, 09 2021. doi: 10.1109/ICECIT54077.2021.9641336.

[73] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. u. Kaiser, and I. Polosukhin. Attention is all you need. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates,

Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[74] C. Wang, P. Nulty, and D. Lillis. A Comparative Study on Word Embeddings in Deep Learning for Text Classification. In *Proceedings of the 4th International Conference on Natural Language Processing and Information Retrieval*, NLPIR '20, pages 37–46, New York, NY, USA, Feb. 2021. Association for Computing Machinery. ISBN 978-1-4503-7760-7. doi: 10.1145/3443279.3443304. URL https://dl.acm.org/doi/10.1145/3443279.3443304.

[75] S. Wang, L. Thompson, and M. Iyyer. Phrase-BERT: Improved phrase embeddings from BERT with an application to corpus exploration. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 10837–10851, Online and Punta Cana, Dominican Republic, Nov. 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.846. URL https://aclanthology.org/2021.emnlp-main.846.

[76] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, Oct. 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL https://aclanthology.org/2020.emnlp-demos.6.

[77] K.-C. Wong. A short survey on data clustering algorithms. In *2015 Second International Conference on Soft Computing and Machine Intelligence (ISCMI)*, pages 64–68, 2015. doi: 10.1109/ISCMI.2015.10.

[78] H. Xu, H. Zhang, and T.-E. Lin. *Discovering New Intents Via Constrained Deep Adaptive Clustering with Cluster Refinement*, pages 99–113. Springer Nature Singapore, Singapore, 2023. ISBN 978-981-99-3885-8. doi: 10.1007/978-981-99-3885-8_8. URL https://doi.org/10.1007/978-981-99-3885-8_8.

[79] H. Yang, Q. Liu, J. Zhang, X. Ding, C. Chen, and L. Wang. Community Detection in Semantic Networks: A Multi-View Approach. *Entropy*, 24(8):1141, Aug. 2022. ISSN 1099-4300. doi: 10.3390/e24081141. URL https://www.mdpi.com/1099-4300/24/8/1141. Number: 8 Publisher: Multidisciplinary Digital Publishing Institute.

[80] H. Zhang, H. Xu, T.-E. Lin, and R. Lyu. Discovering new intents with deep aligned clustering, 2020. URL `https://arxiv.org/abs/2012.08987`.

[81] Y. Zhao and G. Karypis. Criterion Functions for Document Clustering.

[82] Zongqing Lu, Yonggang Wen, and Guohong Cao. Community detection in weighted networks: Algorithms and applications. In *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 179–184, San Diego, CA, Mar. 2013. IEEE. ISBN 978-1-4673-4575-0 978-1-4673-4573-6 978-1-4673-4574-3. doi: 10.1109/PerCom.2013.6526730. URL `http://ieeexplore.ieee.org/document/6526730/`.

A | Additional Results and Discussion

This Appendix contains the results obtained by the experiments with all the hyperparameters explained in Chapter 4. We have collected many outcomes by tuning the hyperparameters regarding the supervised learning experiments (Tables 4.5, 4.6) and the unsupervised learning experiments (Table 4.7). The aim of this Appendix is to provide the best model configurations in order to better compare and analyze them in Chapter 5. In order to assess the optimal model, several outcomes are compared using the NMI and ARI metrics (as explained in Sec. 5.4). To better visualize the best outcomes inside the tables, a gray color is used.

In addition, this Appendix provides analysis of a few experimental patterns that, while not entirely relevant to the thesis, can still be used as a starting point for future investigations.

A.1. Results Supervised Learning Experiments

A.1.1. BCE

The hyperparameters used for the training of the models with $\mathcal{L}_{BCE}$ are described in Table 4.6. For each of the four models (Fig. 4.3), we have three different results due to the different hidden layer size provided (64 - 512 - 1024). All the results obtained by training models with $\mathcal{L}_{BCE}$ and applying Louvain on the output are plotted in Table A.1 and in Table A.2.

Fig. A.1 and A.2 show the results of the experiments, plotting the four evaluation metrics for each model. The aim is to see the presence of patterns varying the hidden layer sizes. The hidden layer sizes are in the x-axis, while on the y-axis we have the metrics scores.

In M^1_{BCE} (Fig. A.1a) with only two layers and a ReLu, we see that all the evaluation metrics scores decrease with the increment of the hidden size. However, in M^2_{BCE} (Fig. A.1b), even if there are only two layers (but a Sigmoid instead of a ReLu), we see that

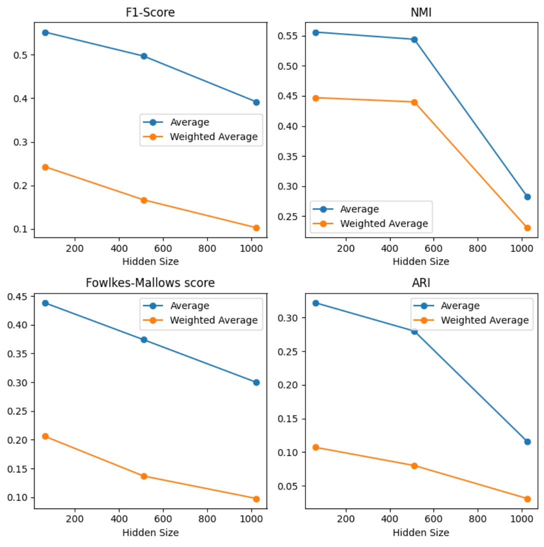
Model Name	**Hidden Size**	**Average of Evaluation Scores**				**Time (s)**
		F1 Score	NMI	ARI	Fowlkes Score	
M^1_{BCE}	64	0.552	0.556	0.322	0.438	8,279.43
	512	0.497	0.544	0.28	0.374	11,936.48
	1024	0.392	0.283	0.116	0.3	10,153.51
M^2_{BCE}	64	0.458	0.472	0.197	0.316	13,510.52
	512	0.5	0.544	0.268	0.364	12,964.79
	1024	0.536	0.566	0.3	0.413	8,054.13
M^3_{BCE}	64	0.389	0.271	0.097	0.299	8,933.85
	512	0.335	0.175	0.039	0.258	**6,672.62**
	1024	0.523	0.551	0.316	0.402	12,864.96
M^4_{BCE}	64	0.508	0.552	0.289	0.394	11,191.33
	512	0.535	0.542	0.292	0.405	9,972.51
	1024	**0.57**	**0.595**	**0.354**	**0.465**	8,747.06

Table A.1: Average of evaluation scores and time for different BCE models.

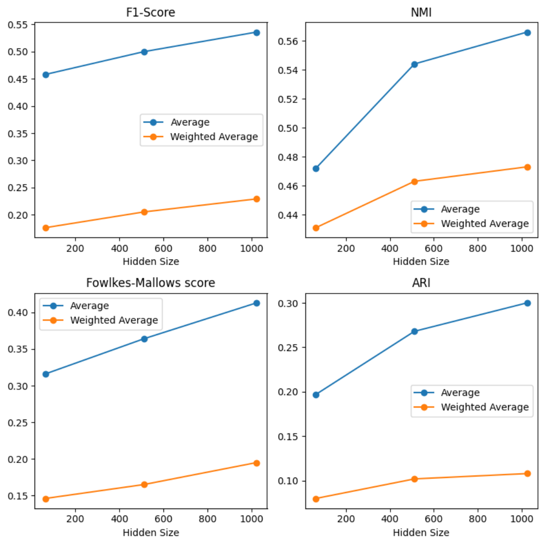
Model Name	**Hidden Size**	**Weighted Average of Evaluation Scores**				**Time (s)**
		F1 Score	NMI	ARI	Fowlkes Score	
M^1_{BCE}	64	**0.243**	0.447	0.107	**0.206**	8,279.43
	512	0.167	0.44	0.08	0.137	11,936.48
	1024	0.103	0.231	0.031	0.098	10,153.51
M^2_{BCE}	64	0.176	0.431	0.08	0.146	13,510.52
	512	0.205	0.463	0.102	0.165	12,964.79
	1024	0.229	**0.473**	0.108	0.195	8,054.13
M^3_{BCE}	64	0.118	0.246	0.031	0.103	8,933.85
	512	0.103	0.227	0.026	0.103	**6,672.62**
	1024	0.15	0.419	0.078	0.125	12,864.96
M^4_{BCE}	64	0.16	0.449	0.081	0.133	11,191.33
	512	0.161	0.427	0.079	0.128	9,972.51
	1024	0.228	0.466	**0.113**	0.192	8,747.06

Table A.2: Weighted average of evaluation scores and time for different BCE models.

the evaluation metrics increase with the increment of the hidden size. This is due for the first activation function which is a Sigmoid. So from Fig. A.1, we can say that when we have a model with two layers and a ReLu activation function after the input, it is better to use a small hidden input layer size hyperparameter. Instead, if we have a Sigmoid, using higher hidden input layer size provides higher scores.

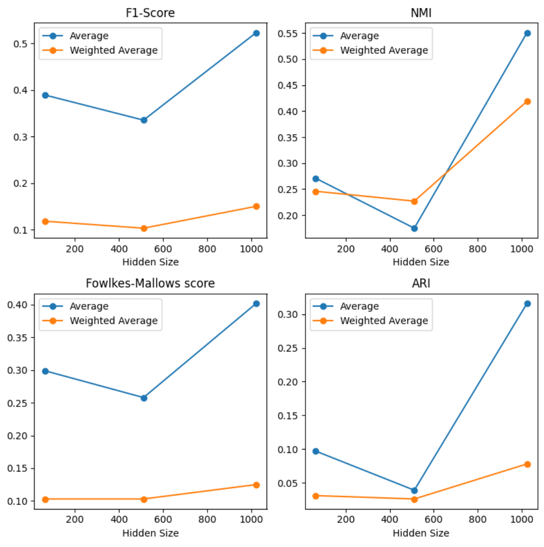


(a) Results of M^1_{BCE}.

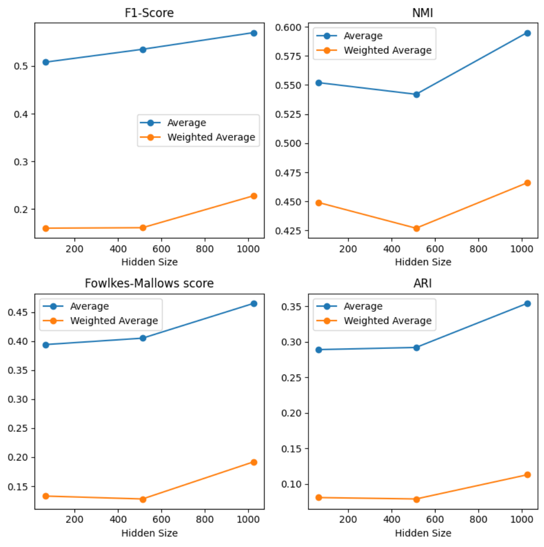


(b) Results of M^2_{BCE}.

Figure A.1: Comparison of two layers models M^1_{BCE} and M^2_{BCE}.



(a) Results of M^3_{BCE}.



(b) Results of M^4_{BCE}.

Figure A.2: Comparison of three layers models M^3_{BCE} and M^4_{BCE}.

Regarding M^3_{BCE} and M^4_{BCE} (Fig. A.2), we notice that in both the models, the scores increase with the increment of the hidden size. We give the model extra parameters by increasing the hidden size, allowing it to learn more complicated representations. This additional capacity may allow the model to better match the training data, leading to greater performance. M^4_{BCE} (Fig. A.2b) is more stable than the third one A.2a, showing that a model with more hidden layers and ReLu as activation functions performs better increasing the hidden input layer size.

A.1.2. Margin Ranking Loss

The hyperparameters used for the models training with $\mathcal{L}_{MRL}$ are described in Table 4.5. For each of the two models M^1_{MRL} and M^2_{MRL} (Fig. 4.2), there are several outcomes due to the different hidden layer size provided (512 - 1024) and different margin γ values. All the results obtained are plotted in Table A.3 and Table A.4. After having trained them, the three community detection algorithms are used on the outputs.

Name	γ	**Hidden Size**	**Algorithm**	**Average of Evaluation Scores**				**Time (s)**
				F1 Score	NMI	ARI	Fowkles Score	
M^1_{MRL}	1	512	Kruskal	0.499	0.761	0.335	0.378	18,788.43
			Louvain	0.241	0.426	0.097	0.225	21,179.56
			Greedy	0.179	0.276	0.048	0.185	3,674.72
		1024	Kruskal	**0.593**	0.65	0.337	0.412	5,944.9
			Louvain	0.513	0.477	0.253	0.381	6,262.78
			Greedy	0.411	0.223	0.1	0.315	1,033.58
M^2_{MRL}	1	512	Kruskal	0.497	0.759	0.335	0.378	17,317.4
			Louvain	0.48	0.432	0.201	0.38	7,792.32
			Greedy	0.388	0.146	0.039	0.324	1,344.06
		1024	Kruskal	0.55	0.676	0.325	0.391	9,979.61
			Louvain	0.466	0.46	0.232	0.356	9,971.64
			Greedy	0.366	0.18	0.062	0.284	1,609.15
	10	512	Kruskal	0.495	0.759	0.331	0.374	16,599.63
			Louvain	0.569	0.479	0.279	**0.44**	3,843.07
			Greedy	0.467	0.189	0.092	0.364	**747.81**
		1024	Kruskal	0.5	**0.762**	**0.338**	0.381	18,117.27
			Louvain	0.456	0.42	0.202	0.326	8,912.74
			Greedy	0.374	0.187	0.064	0.275	1,659.41

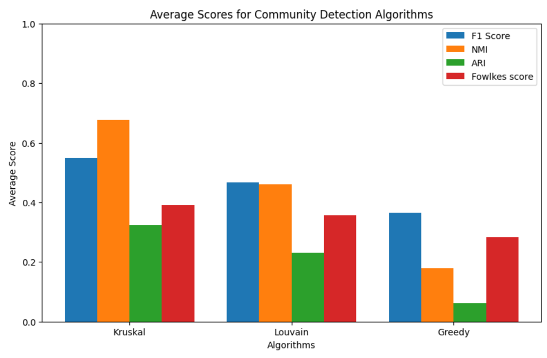
Table A.3: Average of evaluation scores and time for Margin Ranking Loss.

Table A.3 and Table A.4 are characterized by the average and the weighted average of the evaluation scores. In order to have a better visualization of the results, new graphs are provided in the following sections. From our analyses, we can see how the best M^2_{MRL} performs better than the best M^1_{MRL}. The best configuration for M^2_{MRL} is: $\gamma = 10$ - hidden size $= 1024$. Having these values is fundamental to run an analysis and comparison with the best model obtained training using BCE and Margin Ranking Loss, visualized in Chapter 5.

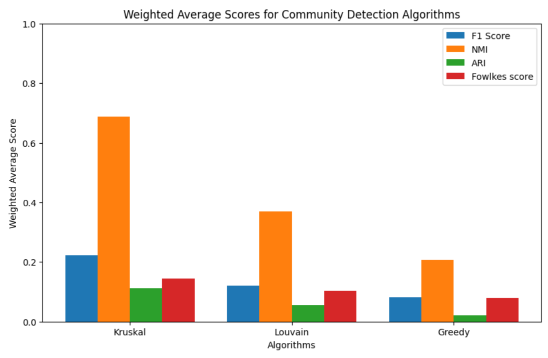
A chart representation of the numerical values in provided in Fig. A.3 and Fig. A.4. The aim is to show the differences among the community detection algorithms. Each color corresponds to a different evaluation metric and the three main areas inside the graphs represent the three different community detection algorithms.

Name	γ	Hidden Size	Algorithm	Weighted Average of Evaluation Scores				Time (s)
				F1 Score	NMI	ARI	Fowkles Score	
M^1_{MRL}	1	512	Kruskal	0.445	0.749	0.293	0.333	18,788.43
			Louvain	0.207	0.408	0.081	0.193	21,179.56
			Greedy	0.151	0.264	0.04	0.158	3,674.72
		1024	Kruskal	0.294	0.697	0.166	0.202	5,944.9
			Louvain	0.182	0.397	0.089	0.156	6,262.78
			Greedy	0.136	0.238	0.043	0.127	1,033.58
M^2_{MRL}	1	512	Kruskal	0.443	0.748	0.293	0.333	17,317.4
			Louvain	0.155	0.367	0.064	0.14	7,792.32
			Greedy	0.115	0.212	0.027	0.115	1,344.06
		1024	Kruskal	0.222	0.687	0.113	0.145	9,979.61
			Louvain	0.12	0.37	0.055	0.103	9,971.64
			Greedy	0.081	0.207	0.021	0.08	1,609.15
	10	512	Kruskal	0.442	0.748	0.29	0.33	16,599.63
			Louvain	0.2	0.405	0.09	0.181	3,843.07
			Greedy	0.147	0.248	0.041	0.147	**747.81**
		1024	Kruskal	**0.447**	**0.771**	**0.296**	**0.336**	18,117.27
			Louvain	0.13	0.351	0.053	0.111	8,912.74
			Greedy	0.098	0.212	0.023	0.094	1,659.41

Table A.4: Weighted average of evaluation scores and time for Margin Ranking Loss.



(a) Average of the metrics on M^2_{MRL} with hidden size 1024.



(b) Weighted Average of the metrics on M^2_{MRL} with hidden size 1024.

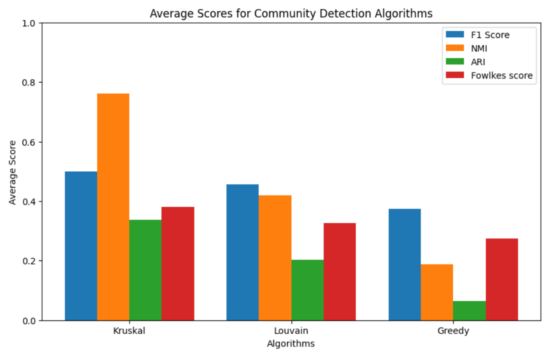
Figure A.3: Results with hidden size 1024 - $\gamma = 1$.

A.1.3. Comparison Community Detection Algorithms

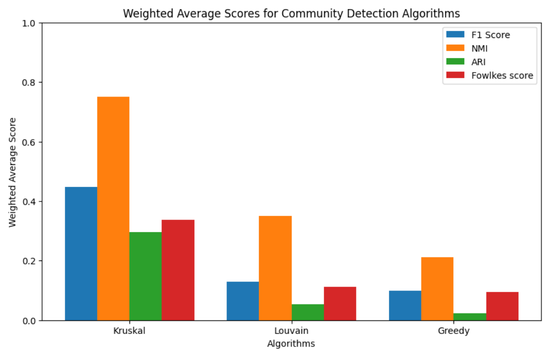
Table A.5 displays the results and differences of the community detection algorithms, applied to M^2_{MRL}. These numerical outcomes are taken from Table A.4.

A.1.4. Comparison BCE - Margin Ranking Loss

In order to do a better comparison between the supervised learning approaches, we compare the best results obtained by the best BCE Model configurations and the best Margin



(a) Average of the metrics on M^2_{MRL} with hidden size 1024.



(b) Weighted Average of the metrics on M^2_{MRL} with hidden size 1024.

Figure A.4: Results with hidden size 1024 $\gamma = 10$.

Method	**F1 Score**		**NMI**	
	Average	**Weighted Average**	**Average**	**Weighted Average**
Kruskal	**0.5**	**0.447**	**0.762**	**0.771**
Louvain	0.456	0.13	0.42	0.351
Greedy	0.374	0.098	0.187	0.212
Method	**ARI**		**Fowlkes score**	
	Average	**Weighted Average**	**Average**	**Weighted Average**
Kruskal	**0.338**	**0.296**	**0.381**	**0.336**
Louvain	0.202	0.053	0.326	0.111
Greedy	0.064	0.023	0.275	0.094

Table A.5: Comparison of community detection algorithms.

Ranking Loss Model. The results are illustrated in Table A.6. We highlight the higher values to make the comparison easier to visualize.

Method	**BCE**		**Margin Ranking Loss**	
	Average	**Weighted Average**	**Average**	**Weighted Average**
F1 Score	0.536	0.15	**0.5**	**0.447**
NMI	0.566	0.473	**0.762**	**0.771**
ARI	0.3	0.108	**0.338**	**0.296**
Fowlkes Score	**0.413**	0.195	0.381	**0.336**
Time (sec)	**8,054.13**		18,117.27	

Table A.6: Comparison between the best BCE and Margin Ranking Loss models.

A.2. Results Unsupervised Learning Experiments

A.2.1. HDBSCAN

The hyperparameters used for computing HDBSCAN are described in Table 4.7. Since there are two hyperparameters `min_cluster_size` α and `min_samples` β that can be tuned with three values each, we obtain in total nine different outcomes, showed in Table A.7 and in Table A.8. The highlighted values are the ones with highest score among all of the HDBSCAN experiments.

Name	α	β	Average of Evaluation Scores				Time (s)
			F1 Score	NMI	ARI	Fowkles Score	
H1		2	**0.52**	0.667	**0.191**	0.259	**612.27**
H2	2	10	0.463	0.656	0.076	0.121	1,380.89
H3		200	0.377	0.667	0.007	0.028	2,274.66
H4		2	0.491	0.642	0.111	0.167	1,554.36
H5	10	10	0.445	0.653	0.057	0.089	1,276.49
H6		200	0.325	**0.701**	0.002	0.701	891.15
H7		2	0.347	0.685	0.004	0.011	804.76
H8	200	10	0.336	0.694	0.003	0.006	890.65
H9		200	0.317	**0.703**	0.002	0.005	885.01

Table A.7: Average of evaluation scores and time for HDBSCAN.

Name	α	β	Weighted Average of Evaluation Scores				Time (s)
			F1 Score	NMI	ARI	Fowkles Score	
H1		2	0.369	0.702	0.091	0.124	**612.27**
H2	2	10	0.422	0.705	0.073	0.103	1,360.89
H3		200	0.365	0.714	0.012	0.036	2,274.66
H4		2	0.444	0.693	**0.112**	**0.143**	1,554.36
H5	10	10	**0.446**	0.686	0.07	0.105	1,276.49
H6		200	0.284	0.77	0.005	0.01	891.15
H7		2	0.353	0.718	0.013	0.036	804.76
H8	200	10	0.32	0.746	0.009	0.022	890.65
H9		200	0.257	**0.778**	0.003	0.002	885.01

Table A.8: Weighted average of evaluation scores and time for HDBSCAN.

Table A.8 shows that we have higher NMI values with $\alpha = 200$ and $\beta = 200$. Therefore this configuration is used to compare it with the K-Means outcomes. Table A.9 illustrates the results obtained by the best HDBSCAN configuration so far.

In order to better find any pattern among the evaluation metrics, we plot some results in Fig. A.5 and Fig. A.6. Every plot contains the results with the another hyperparameter β, showed on the x-axis, while on the y-axis the evaluation metrics scores are displayed.

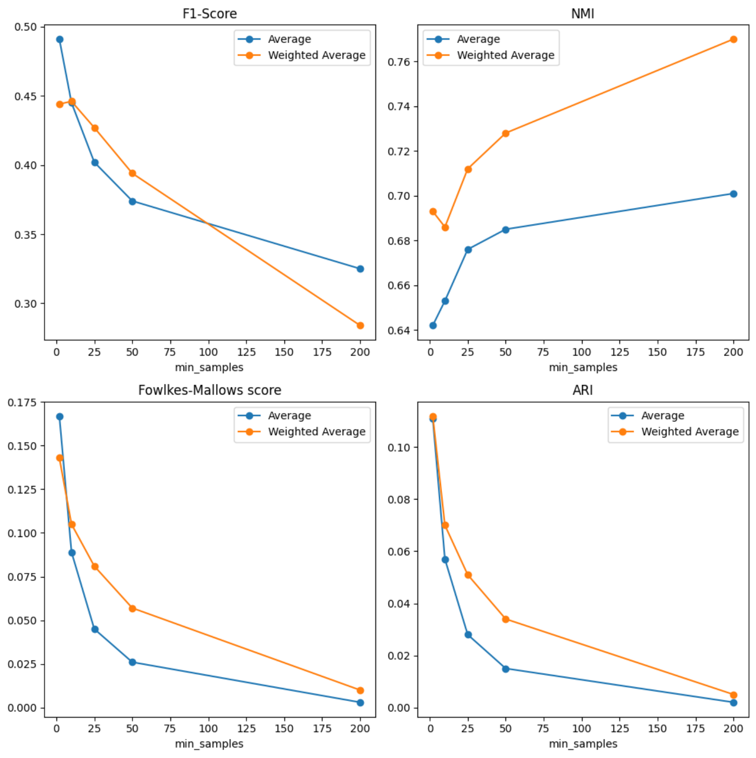


Figure A.5: HDBSCAN with $\alpha = 10$.

With Fig. A.5 and Fig. A.6, some patterns among the evaluation metrics can be identified. Increasing the β value has different effects on various evaluation metrics:

- **F1 Score**: it *decreases* with the increase of the β value.
- **NMI**: it *increases* with the increase of the β value.
- **ARI**: it *decreases* with the increase of the β value.
- **Fowlkes-Mallows score**: it *decreases* with the increase of the β value.

A.2.2. K-Means with Elbow Method

In Table A.10, we report the results from K-Means algorithm.

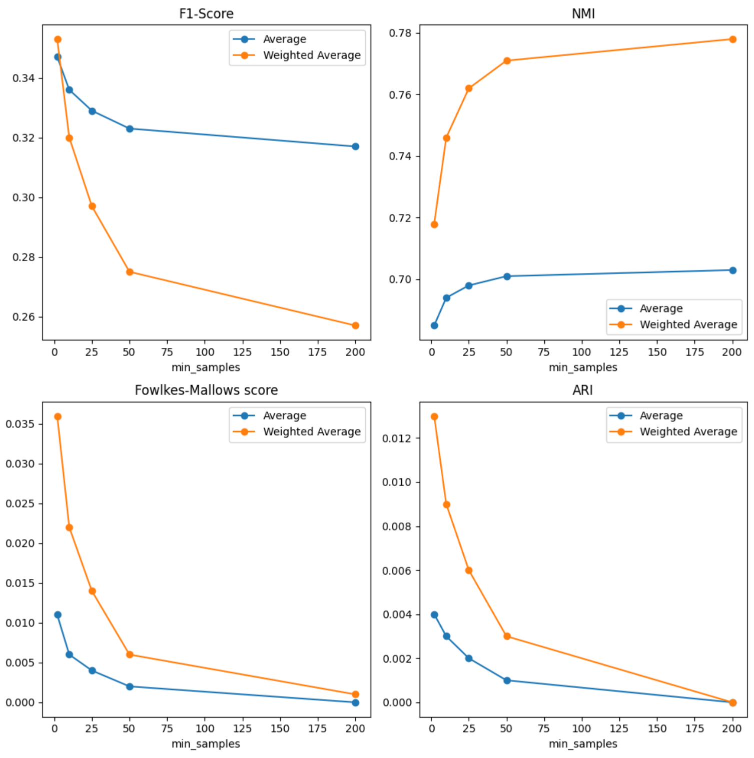


Figure A.6: HDBSCAN with $\alpha = 200$.

Metric	Average	Weighted Average
F1 Score	0.491	0.444
NMI	0.642	0.693
ARI	0.111	0.112
Fowlkes score	0.167	0.143
Time (sec)	1,554.36	

Table A.9: Evaluation results with the best HDBSCAN configuration ($\alpha = 10$ - $\beta = 2$).

Metric	Average	Weighted Average
F1 Score	0.741	0.59
NMI	0.627	0.423
ARI	0.46	0.234
Fowlkes score	0.455	0.442
Time (sec)	6,128.37	

Table A.10: Evaluation results with K-Means.

A.2.3. Comparison HDBSCAN - K-Means

In order to do a better comparison between these two algorithms, we compare the best results obtained by the best HDBSCAN configurations and K-Means. The results are illustrated in Table A.11. We highlight the higher values to make the comparison easier to visualize.

Method	**HDBSCAN**		**K-Means**	
	Average	**Weighted Average**	**Average**	**Weighted Average**
F1 Score	0.491	0.444	**0.741**	**0.59**
NMI	0.642	**0.693**	0.627	0.423
ARI	0.111	0.112	**0.46**	**0.234**
Fowlkes Score	0.167	0.143	**0.455**	**0.442**
Time (sec)	**1,554.36**		6,128.37	

Table A.11: Comparison between HDBSCAN and K-Means

A.3. Comparison Supervised and Unsupervised Learning

This final comparison helps to show the main differences between the outcomes from supervised and unsupervised learning. We pick the best model from each algorithm to run the final analysis. Table A.12 illustrates the results.

Metric	M^2_{BCE}		M^2_{MRL}		**HDBSCAN**		**K-Means**	
	Avg	**W. Avg**	**Avg**	**W. Avg**	**Avg**	**W. Avg**	**Avg**	**W. Avg**
F1 Score	0.536	0.229	0.5	0.447	0.491	0.444	**0.741**	**0.59**
NMI	0.566	0.473	**0.762**	0.771	0.642	**0.693**	0.627	0.423
ARI	0.3	0.108	0.338	**0.296**	0.111	0.112	**0.46**	0.234
Fowlkes score	0.413	0.195	0.381	0.336	0.167	0.143	**0.455**	**0.442**
Time (s)	8,054.13		18,117.27		**1,554.36**		6,128.37	

Table A.12: Comparison of best metric values for different algorithms.

Table A.12 is used to run the final analysis in Table 5.4.

List of Figures

List of Tables

Acknowledgements

I would like to thank Gavagai for allowing me to write the thesis with them and for offering resources for study. I would like to thank my supervisor, Filip Cornell, for his patience and wisdom who helped me a lot over the last few months, and for believing in me. I would like to thank Annick for always encouraging me and pushing me to give my best. I would like to thank my parents, Rossella and Marcello, my family and my friends for their support and encouragement. I could not have completed this project without their support. I would like to thank Federico Schiepatti for his support over the past few years, being always available for every doubt. At the end, I would like to spend a few words on my grandmother who recently passed away. Thank you for your support, thank you for raising me and thank you for always being there for me. You will always be next to me. Ciao Nonna, ti voglio bene.