POLITECNICO DI TORINO

Master Degree course in Computer Engineering

Master Degree Thesis

Clustering feedback data with Logistic Network Lasso

Supervisors
prof. Barbara Caputo
prof. Alex Jung (Aalto University)

Candidate
Simone Desogus

Accademic year 2017-2020

To my father who could not make it to see this journey finish,

To my mother who supported me with endless patience,

To my brother who guided me from the beginning,

To my partner who has been my sunshine in the murky days.

Summary

Feedback gathered from university courses' are the most effective tool given to students to shape lectures according to their needs and difficulties. Nonetheless, feedback is often neglected by both students and teachers, even if studies show how feedback can improve the quality of the student's experience as well as being an essential tool for the teacher's development.

With the improvement in Natural Language Processing and the Machine Learning field, it is now possible to analyze text and effortlessly gather insights. These insights ease the professor's work when faced with the task of reading feedback which, depending on the course, can be thousands. The Logistic Network Lasso is a novel semi-supervised machine learning algorithm that has been applied in the binary classification scenario and compared to algorithms such as Maxflow and Belief Propagation, resulting in overall increased accuracy.

This thesis aims to answer two research questions in order to further analyze the capabilities of the Logistic Network Lasso algorithm.

The first research question wanted to find the most efficient way to transform the feedback data into suitable embeddings for Logistic Network Lasso. Various standard clustering algorithms such as K-Means, Affinity Propagation, Spectral Clustering and DBSCAN are compared using different embedded vectors. This thesis gives a general overview of the most common embedding techniques, which are Word2Vec, Doc2Vec, GloVe, FastText and BERT. From the set of embedding techniques studied, GloVe, FastText and BERT were found to be superior in terms of the clustering performance metrics analyzed when used with standard clustering algorithms without Logistic Network Lasso. The clustering performance metrics that were analyzed in this thesis are divided in two classes, clustering performance metrics that rely on external indexes and clustering performance metrics that depend on internal indexes. The metrics used for the first class are Rand-Index, Adjusted Rand-Index, Normalized Mutual Information, Adjusted Mutual Information and Fowlkes-Mallows score. The metrics used for the second class are the Silhouette Coefficient, Davies-Boulding Index and the Calinski-Harabasz index. The metrics for the second class were found to be unreliable when assessing the performance of the clustering methods when the BERT embeddings were used, probably due to the higher dimensionality of the feature vectors.

The impact of the conventional pre-processing techniques in this clustering scenario has been investigated. Surprisingly, standard NLP pre-processing approaches such as stemming affected the metrics negatively and were removed from the pre-processing pipeline. Moreover, another standard pre-processing technique known as stopwords removal was not applied, because the semantic of the sentences had to be preserved.

The second research question wanted to find if the Logistic Network Lasso is an optimal choice for tackling the binary clustering feedback problem. In order to answer the research question, standard clustering algorithms were compared with the Logistic Network Lasso incorporating the best embedding techniques discovered during the first research question.

Logistic Network Lasso requires weights for the edges between data points that are considered similar. BERT similarity scores were an adequate candidate and improved the performance of Logistic Network Lasso in binary clustering. Logistic Network Lasso is an algorithm that uses a percentage of the true labels during training. For this reason, a certain improvement over the other unsupervised algorithms tested was expected, proportional to the percentage of true labels used. Using a percentage of 20% of true labels, lnLasso increased the best value of Adjusted

Rand-Index, Normalized Mutual Information and Adjusted Mutual Information of up to 30%, but decreased Rand-Index and Fowlkes-Mallows index. The best combination of embeddings method to use in conjunction with Logistic Network Lasso is either FastText or BERT using the average of both second-last and last-layer. The results of the experiments indicate that lnLasso is a valid choice for binary clustering feedback data.

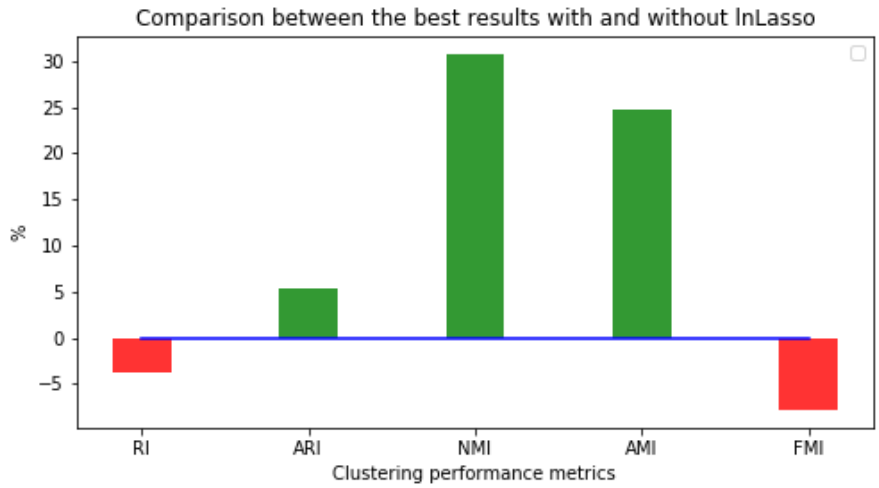

Figure 1. Bar plot showing the increment or decrement in percentage between the best result for each clustering performance metrics, comparing the result obtained with and without lnLasso.

This thesis, for the first time, researched the applicability of the Logistic Network Lasso when binary-clustering feedback data. More research has to be done in the non-binary or multiclass classification scenario, comparing the Logistic Network Lasso with soft clustering techniques. In the non-binary classification scenario, the classes can be selected using topic modelling, automating the process even more for understanding the feedback.

Contents

Abbreviations

lnLasso	Logistic Network Lasso
NLP	Natural Language Processing
Lasso	Least Absolute Shrinkage and Selection Operator
MLBP	Machine Learning Basic Principles
EDM	Educational Data Mining
BOW	Bag Of Words
CBOW	Continuous Bag Of Words
GloVe	Global Vectors
LSA	Latent Semantic Analysis
RNN	Recurrent Neural Networks
GLUE	General Language Understanding Evaluation
NLU	Natural Language Understanding
BERT	Bidirectional Encoder Representations from Transformers
RoBERTa	Robust Optimized BERT pretraining Approach
ALBERT	A Lite BERT
LSTM	Long Short-Term Memory
SICK	Sentence Involving Compositional Knowledge
PV-DM	Distributed Memory version of Paragraph Vector
PV-DBOW	Distributed Bag of Words version of Paragraph Vector
PPDB	Paraphrase Database
ELMo	Embeddings from Language Models
ULMFiT	Universal Language Model Fine-Tuning
SNLI	Stanford Natural Language Inference Corpus
MLM	Masked Language Model
GPT	Generative Pre-trained Transformer
MRPC	Microsoft Research Paraphrase Corpus
STS-B	Semantic Textual Similarity Benchmark
RBF	Radial Basis Function
DBSCAN	Density-based spatial clustering of applications with noise
PCA	Principal Component Analysis
TV	Total Variation
ADMM	Alternating Direction Method of Multipliers
QA	Question Answering
NLI	Natural Language Inference
NSP	Next Sentence Prediction
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
RI	Rand-Index
ARI	Adjusted Rand-Index
MI	Mutual Information
NMI	Normalized Mutual Information
AMI	Adjusted Mutual Information
FMI	Fowlkes-Mallows score
SC	Silhouette Coefficient
DBI	Davies-Bouldin Index
CBI	Calinski-Harabasz Index
AP	Affinity Propagation
SPC	Spectral Clustering
DBCV	Density-Based Clustering Validation Index
DSC	Density Sparseness of a Cluster
DSPC	Density Separation of a Pair of Cluster

Chapter 1

Introduction

This section describes the challenges of clustering open-ended feedback, and presents the research questions that will be answered in the rest of the thesis.

1.1 Overview

Learning Analytics applications are becoming extensively more researched, and they are considered as "*the most dramatic factor shaping the future of higher education*" [1]. Thanks to the increased popularity of Educational Data Mining (EDM), we have seen how these applications can improve the quality of the student experience [2], how these insights can be used to improve e-learning systems [3] and how you can even improve student performance [4].

Feedback gathered from university courses' are one of the tools that can be used to gather insights on how well a course is perceived by students. Feedback are usually divided between open-ended and closed-ended questions. Feedback questionnaires are composed of both of these types of questions. Closed-ended question produce feedback following a structure, that enables them to be easily analyzed with little effort. Open-ended questions on the other hand are more difficult to analyze in an automated way, since they are written in a natural language, i.e. English. Open-ended questions are useful to understand student's feelings or attitudes and this information is often lost or not well depicted by closed-ended questions. The rest of this thesis focuses only on open-ended feedback and will only be referred to as "feedback".

Feedback can often say the same concept using different words, because two students can think in different ways. This phenomenon increases the difficulty of clustering similar feedback together, because the tool used to cluster the feedback has to understand the semantics of the sentences, which is a difficult task. In some cases, the huge amount of feedback discourages the teacher reading them because the amount of work is too extensive for the teacher to handle, and the feedback may be skimmed instead of carefully read, and useful information provided by the students will be lost.

Whether feedback is the most effective and reliable method for teacher's evaluation has been studied [5], and two-thirds of the teachers and students that were involved in this study agreed that student's feedback is an important tool for educators development. The study involved teachers and students from a medical college. The educator's performance was evaluated after a three month teaching period followed by three months where teachers could hone their skills in a precise way thanks to the feedback that guided them toward the students needs. Following another three months the educator's performance and the feedback's effectiveness was evaluated. The student's feedback focused on the teacher's planning and preparation for the lecture, classroom environment and after class consultation availability. This feedback has been found to be reliable, and teachers agreed on its effectiveness.

Natural Language Processing (NLP) studies the interaction between computers and the natural languages, how to analyze this data and use it in our daily lives. Some applications of NLP is

Machine Translation which is used in Google Translate, Question Answering used by bots to help the users find the correct Frequently Asked Question (FAQ) and Speech Recognition used for the automatic generation of subtitles in videos. The NLP applications are only briefly introduced here. Thanks to the progress made in this field we can now efficiently deal with vast quantities of text and analyze it, therefore provide real-time statistics, that can be used in parallel with the feedback, modifying a course accordingly.

1.2 Related research

Most of the research that has been done using the Least Absolute Shrinkage and Selection Operator (Lasso) to network structured data (nLasso) used the squared error as the loss function. Recently there has been research on binary classification and clustering problems with network structured data that used the logistic loss function instead, and the name Logistic Network Lasso (lnLasso) was coined. The research on lnLasso mostly focused on clustering and binary classification [6], and learning networked exponential families [7]. The research compared the lnLasso to other algorithms such as Maxflow and Belief Propagation, showing that lnLasso was overall better in terms of accuracy after a certain number of iterations.

1.3 Problem statement

The aim of this thesis is to find out if the lnLasso algorithm can be used to tackle the problem of clustering University courses' feedback, comparing it with other modern and standard solutions. University courses' organization can vary, and with that the amount of feedback generated by students. University courses' can be divided into parts and for every part there could be a feedback questionnaire. Dividing the course into parts enables the teacher to tackle problems quickly, changing the material and the lesson's structure accordingly to what the students may or may not find difficult to understand. This type of feedback questionnaire divided into parts is the case for the dataset utilized in this thesis, as shown in paragraph Dataset.

Clustering the feedback together is an important step to save time for the teacher, especially when the amount of feedback becomes too time-consuming for a single person to analyze in a short time, i.e. the case depicted in the previous paragraph. Clustering allows the teacher to look at the main trends and problems that student have, and quickly act accordingly, making the students more involved in shaping the lectures.

This problem is not limited to the universities or eLearning environments, but can be extended to any scenario where a considerate amount of written feedback is involved, like companies delivering a product to a user, healthcare research, events or any scientific study that requires open-ended questions.

In order to solve the research question of if the lnLasso algorithm can be a competitive clustering technique for educational data, another research question arises. What is the most effective way to capture the features of the educational data into a model?

1.4 Organization of the thesis

This thesis is organized into five main chapters. After the introduction, the second chapter describes some of the many techniques used in NLP to obtain sentence embeddings which are Word2Vec, Doc2Vec, FastText, GloVe, Recurrent Neural Networks, Long Short-Term Memory neural networks, Skip-Thought Vectors, Paragram-Phrase Embeddings, Transformers, ELMo, ULMFiT, OpenAI GPT and BERT.

The third chapter describes the dataset used for this thesis, the pre-processing techniques that are usually used when solving a NLP problem, and some benchmarks that were used to compare and train the algorithms. This chapter continues describing how to model a machine learning

problem and describes the clustering techniques that will be analyzed in the next chapter of the thesis, such as K-Means, Affinity Propagation, Spectral Clustering and DBSCAN. The chapter continues describing the clustering evaluation metrics that will be used to compare the different clustering algorithms with the lnLasso, dividing them in metrics that rely on external indexes and metrics that depend only on internal indexes. The third chapter ends with describing the lnLasso algorithm.

The fourth chapter describes the configuration in which the results were obtained and describes the different experiments. The first round of experiments compares the different clustering algorithms using the first three rounds of the datasets, pruning diverse configurations when results are not reliable. The second round of experiments compares the distinct clustering algorithms using diverse embeddings with and without stemming as a pre-processing step. The third round of experiments compares the different clustering algorithms using the last three rounds of the dataset, with diverse embeddings. The fourth round of experiments compares which of the distinct embeddings and configuration of lnLasso provides the best results, without a weighting schema. The fifth and last round of experiments uses the information from the previous round to select the top three embeddings and configurations for lnLasso and further analyzes the algorithm using as the weighting schema the BERT similarity scores.

The fifth and last chapter draws the conclusions and further expands some questions for future work.

Chapter 2

Background

In order to apply lnLasso, the dataset (see Table 3.1) needs to be represented by an undirected weighted graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathrm{W})$. A particular node $i \in \mathcal{V}$ of the graph represents a feedback encoded as a phrase vector. Due to the fact that similar feedback are clustered together, the feedback's semantic meaning must be preserved. In the literature, there are many ways of encoding words or phrases as vectors, and this chapter aims at describing a part of them.

2.1 Encoding feedback

Text can be unpredictable and machine learning algorithms prefer well-defined vectors of numbers, therefore text has to be converted into numbers that represent the text properties. This step is called feature extraction or feature encoding. The easiest way to encode feedback into numeric vectors is to use one-hot encoding. Given a vocabulary of words that consists of all the distinct words used in the feedback, the feedback will be truncated into words first and words will be converted into vectors that consists of all zeros and only ones where the index of the word in the vocabulary is. This will generate very sparse vectors, that do not carry the meaning of words. For this reason, one-hot encoding is not suitable to be selected as a technique to encode feedbacks for the scope of this thesis.

2.1.1 Word2Vec

Log-linear models "Continuous Bag of Words" (CBOW) or the similar "Skip-gram" first introduced in [8], also known as the word2vec models, are widely adopted thanks to their simplicity and flexibility. When word2vec models were first introduced, for the word similarity task state-of-the-art NLP techniques where more complex non-linear neural networks that did not focus on preserving the similarity between words and represented them as *indices in a vocabulary*, similar to what one-hot encoding does. CBOW and Skip-gram achieved the result of obtaining high quality and high dimensional word vectors using simpler architectures trained on more massive datasets, and instead of focusing only on the scalar distance between words, multiple degrees of similarity between words are preserved [9].

A BOW is a method to represent a text using the occurrence of words in a document, using a vocabulary of unique words present in the input, and an array to keep track of the presence of known words. It is called "Bag" of words because the order of the words in the text is lost. The word2vec model trains a simple neural network with a single hidden layer, but instead of using the neural network itself, it extracts the weights of the hidden layer, which are used to generate the word vectors. The word2vec model is trained by attempting to guess the output (target word) using the neighbour words as input (context words) or vice versa. More precisely, the CBOW architecture tries to predict a word given its context, while Skip-gram tries to predict the context given a word, as shown in Figure 2.2.

In order to train the CBOW model, it needs a tokenized document (see Figure 3.1) as training input. This document is going to be divided into training sample which consists of word pairs (see Figure 2.1). The numbers of neighbours to be considered as word pair is defined by the training window, a parameter of the model which usually is set to five. Five as the dimension of the window means five words preceding the input word and five words after the input word are considered as neighbours. The network is going to learn statistics from how often each pair of words shows up in the same window, since the model assumes that words with similar meanings are used nearby in different sentences. By doing this, words with similar meaning will end up with a comparable numeric representation once encoded, so the word "United" will be close to "America" and "Kingdom", and far from "Coconut".

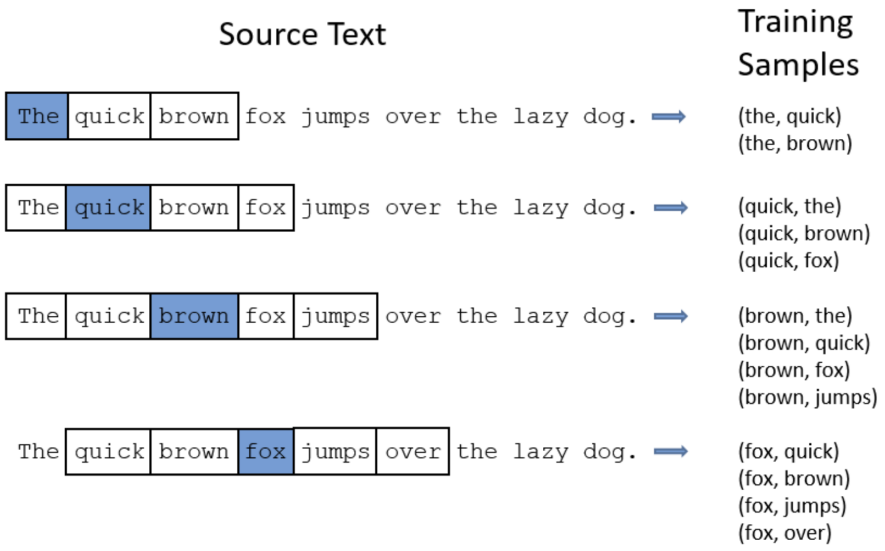

Figure 2.1. Example on how word2vec splits the source text in training samples, with a moving window w = 2.

The difference of architecture between CBOW and Skip-gram (see Figure 2.2) is important when our interest is to give more weight to the rare words. Given the context "Tomorrow will be a really [...] day", CBOW will try to fit words like "beautiful" or "nice". Words like "delightful" since are rarer, will be used less from the model since it is trying to predict the most probable word. Skip-gram instead starts from the word "delightful", and adding to the context the pair will be treated as a new observation. According to the authors of the paper [8] Skip-gram works well when the training data and represents well even rare words, while CBOW is faster to train and has slightly better accuracy for the more frequent words.

CBOW and Skip-gram obtained high scores in the tasks they were tested for but they were limited to single-word recognition, and compound words such as "Air Canada" were treated as two different single words. Another problem was that frequent words like "the" were considered similar to any word, because they were often used together with almost any word. Some improvements were introduced in [10] that addressed these problems. They increased the quality of the words vector and introduced phrase vectors while decreasing the training time.

One approach to encoding feedback using CBOW and Skip-gram could be using the weighted average of word vectors, but by doing this since as explained before the word order is not preserved by the models, two feedback with different meaning that use the same words will be considered as very similar. Another weakness of these two models is that they do not preserve the meaning of words but only the degree of similarity between them.

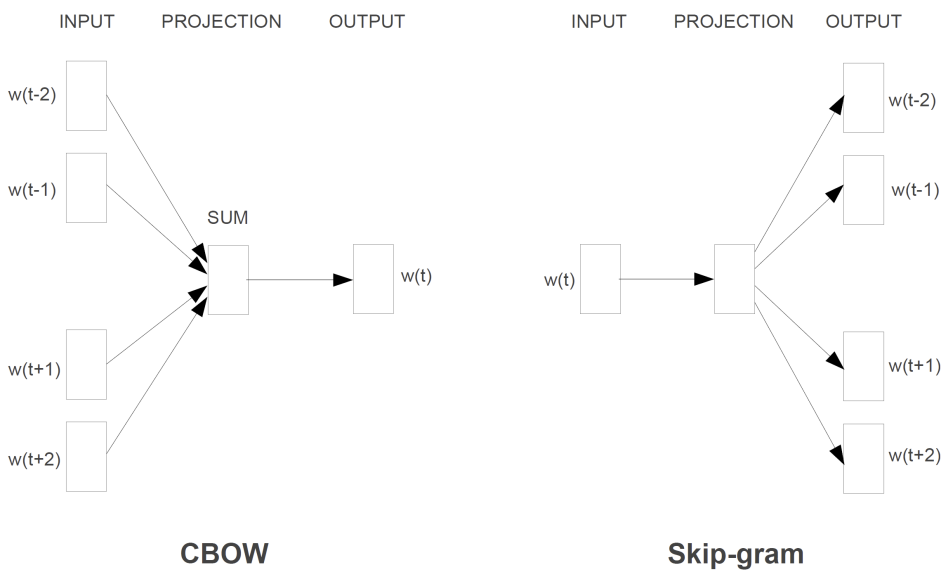

Figure 2.2. CBOW and Skip-gram architectures. CBOW predicts the current word based on the context while Skip-gram predicts surrounding words given the current word. (Figure taken from paper [8])

2.1.2 Doc2Vec

The algorithm Paragraph Vector also known as doc2vec [11] on the other hand, heavily inspired by CBOW and Skip-gram, can represent texts with variable length ranging from sentences to documents, while preserving their meaning. Doc2vec is an extension of Word2vec that adds another vector called Paragraph ID, and it comes in two different variants.

The Distributed Memory version of Paragraph Vector (PV-DM), which uses this document-unique Paragraph vector during the prediction of the next word on top of the context. During the training this Paragraph is trained as well, and after the training it will hold a numeric representation of the document.

The other variant is the Distributed Bag of Words version of Paragraph Vector (PV-DBOW) which is comparable to Skip-gram. From the Paragraph vector the algorithm tries to guess the context. This algorithm is faster and consumes less memory because it doesn't save the word arrays. The authors' of the paper recommend to use a combination of both algorithms, even if the PV-DM alone is enough to reach the state-of-the-art results.

While presenting good results in the paper, others have struggled to reproduce those results [12]. Even if some effort was made to improve the Paragraph Vector by using pre-trained word vectors, doc2vec, in general, performs strongly mostly on very large documents, which is not the case of this thesis' dataset (See Table 3.1).

2.1.3 FastText

FastText is an algorithm [13] created by the Facebook Research Team for learning word embeddings in the context of text classification. FastText takes inspiration from the Word2Vec model [10], but it has some key differences. The main difference with the word2vec model is that instead of using words as the smallest unit to compute the embeddings, words are split into n-grams e.g. the word "eagle" will be split into "[eag, eagl, eagle, agle, gle]". N-grams provide FastText with some advantages over word2vec. Rare words embeddings are easier to compute, because their n-grams might be shared with the representation of common words. If words are not present in the dictionary, they can be split into n-grams and again see if there is a word in the vocabulary

that shares the n-grams. Another advantage is that n-grams perform better in terms of accuracy on smaller datasets compared to Word2Vec and unlike Word2Vec embeddings, the FastText word embeddings can be averaged to form useful sentence embeddings.

Other than this difference, the FastText architecture is very similar to the Word2vec. It has a weight matrix that works as a look-up table for words and a classifier is used for generating the text representation.

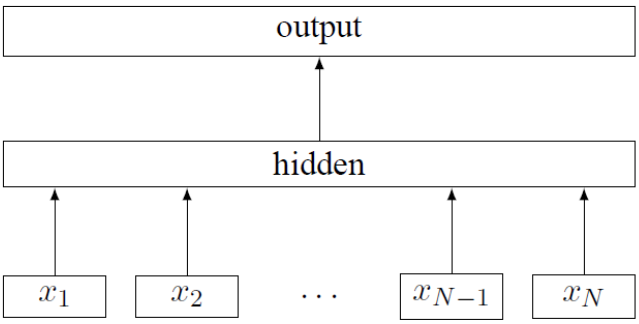

Figure 2.3. Diagram showing the FastText model architecture. The architecture is very similar to the CBOW architecture, Figure 2.2 (Figure taken from paper [13])

One of the goals of FastText is to be efficient with large datasets and in order to do so it uses a hierarchical classifier [14] in which the different classes are organized into a tree based on the Huffman Coding Tree. Exploiting the properties of the Huffman Tree, for uncommon words the FastText model can generate smaller trees, increasing the efficiency. Word2Vec models do not take into account the word order because it is computationally expensive, limiting its efficiency when averaging the word embeddings into a sentence embeddings. FastText overcomes this problem by using a bag of n-grams that will partially encode the word order.

2.1.4 GloVe

Global Vectors (GloVe) [15] are word embeddings that overcome the drawbacks of the word2vec models and Latent Semantic Analysis (LSA). LSA is a technique used in summarizing text and Topic modelling, that uses low-rank approximations to decompose large matrices. LSA is able to leverage statistical information but unable to understand word analogies. Word2vec models are able to do the opposite, understand word analogies using various degree of similarity between words, but they poorly utilize the statistics that are present in the corpus because they train using windows, instead of using global occurrence counts.

GloVe architecture uses a co-occurrence matrix computed using a fixed window size with the same assumption of Word2Vec models, that the co-occurence ratios between words in a context are connected to their meaning, but it also uses global count statistics. Some words are more frequent than others and tend to be noisy, hence occurrences are weighted with a formula $weight(x) = min(1, (x/x_{max})^{\frac{3}{4}})$

Empirical results published in the GloVe paper [15] show that GloVe embeddings are generated relatively fast, achieving better accuracy than the other models. Usually, there is no need to train the vectors because they are downloaded already pre-trained. Hence the speed advantage is not relevant, and further testing in other NLP tasks showed that word2vec and GloVe embeddings perform more or less with the same level of accuracy.

Due to its architecture, this approach understands only word-level information, while to grasp the semantic of a sentence it has to be encoded higher-level information. Even if there are limitations to its architecture, GloVe vectors are still used today as part of more significant and more complex NLP models.

2.1.5 Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a class of neural networks that are heavily used in NLP since they are a natural approach for sequence modelling tasks. RNN have the flexibility to work with sentences of different lengths, something which cannot be achieved in a standard neural network due to its fixed structure. RNN are a type of neural network where the output from the previous iteration is fed as input to the current step, which allows information to persist, having some sort of "memory".

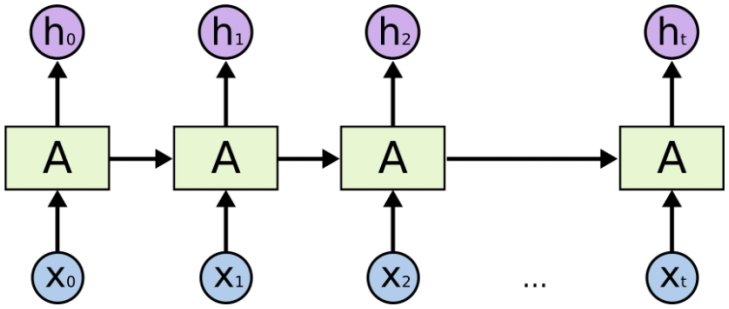

Figure 2.4. Structure of a Recurrent Neural Network. A RNN can be seen as multiple copies of the same simple network, each passing a message to the next network. (Source of the image link)

Using parse trees and RNN [16] has some advantages over word2vec models but since it relies on parse trees, it works only with one sentence while in the same feedback multiple sentences can be present. Another limitation of RNN is having only one direction due to its architecture, and this is translated into assuming the word coming after has no effect on the meaning of the word that is before, which is often not true in natural languages. RNN works with the same assumption made by Word2Vec and GloVe, that in order to predict a word based on the context, the information needed to predict the word is contained in the nearby words. In more complex sentences, the information needed to predict the word is contained at the beginning of the phrase, far away from the last node and outside the window of the Word2Vec model.

By construction, RNN give less weight to the nodes that are too far away and this is known as the "vanishing gradient problem" [17]. In other words, RNN are unable to learn long-term dependencies, which might be the case in the thesis' dataset (See Table 3.1) where for example, feedback number five can be related to feedback number 602, if feedback are used as single units for the input of the RNN model.

2.1.6 Long Short-Term Memory neural networks

More complex architectures that use neural networks such as Long Short-Term Memory (LSTM) [18] have their advantages. LSTMs overcome the weakness of RNNs, thanks to their ability to preserve information over time. LSTMs contain information using a gated cell that acts as a memory cell in a computer. This information can be stored, written, read or erased, using gates that open and close. The difference between a gated cell and a memory cell is, while information on a computer's memory cell is digital consisting of 0 and 1, memory in the gated cells is implemented with element-wise multiplication of sigmoids, that are differentiable and therefore suitable for backpropagation.

As shown in figure 2.5 the gates allow to pass or block information based on its weights. These weights are similar to the ones that regulate input and hidden states, and they are modified in the neural network learning process via making guesses, backpropagation errors and gradient descent.

A tree-structured LSTM called "Tree-LSTM" [19], has been proved to outperforms the chain-structured LSTM in the task of semantic relatedness. Even common variation of LSTMs such as

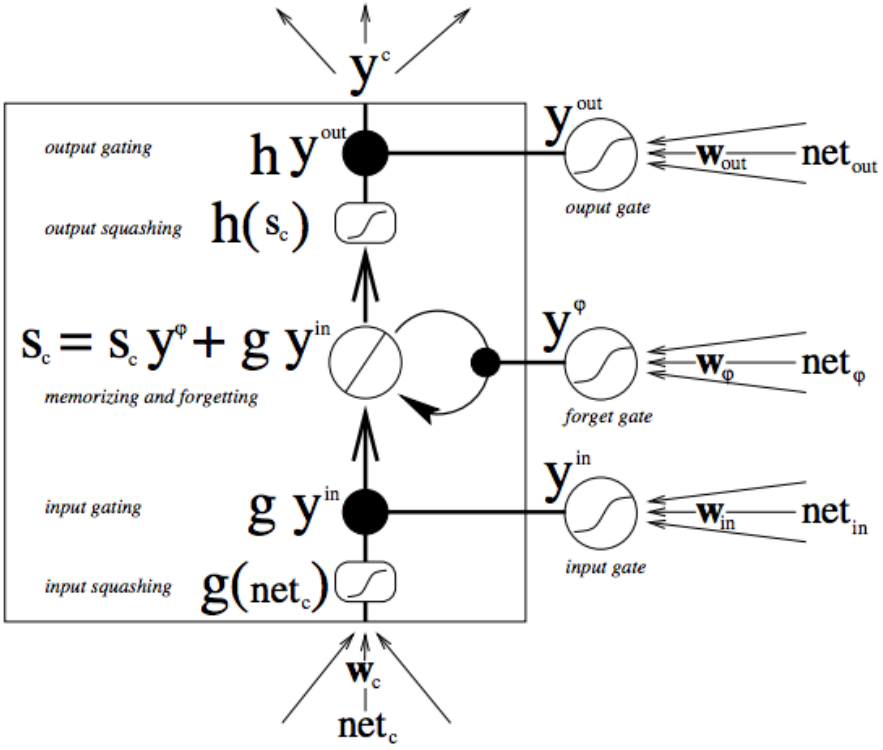

Figure 2.5. Structure of a memory cell in a LSTM. The gates decide how the data flows inside the memory cell. (Source of the image link)

Bidirectional LSTM and Multilayer LSTM [20] are limited by their chain-structure, that allows strictly sequential information propagation. Tree-LSTMs on the other hand, are a type of network topology where each LSTM unit can hold information from various child units. The difference between standard LSTM and Tree-LSTM is that the Tree-LSTM's gating vectors and memory cell updates are dependant on the states of many child units, and instead of having one forget gate, there is a forget gate for each child. This allows to choose how to incorporate information from each child, and to give more or less weight to different children. Another result is that Tree-LSTM are able to outperform heavy feature engineered systems, requiring supervision only at the root of the tree.

Unfortunately, these more complex architectures are powerful with in-domain data, while simpler architectures work better for learning general-purpose embeddings [21].

2.1.7 Skip-Thought Vectors

Skip-Thought Vectors [22] is an unsupervised neural network model for learning representation of sentences in natural languages. These representations have a fixed length, and this is needed because all sentences are replaced with a vector of numbers, and having the same length will make the process of training and understanding natural languages easier.

The Skip-Thought model has three parts (see Figure 2.6):

- **Encoder Network**: A RNN that takes the sentence $x(i)$ at index i and generates the fixed length representation $z(i)$
- **Previous Decoder Network**: A RNN that given the embedding $z(i)$ tries to generate the sentence $x(i-1)$
- **Next Decoder Network**: A RNN that takes the embedding $z(i)$ and tries to generate the sentence $x(i+1)$

The Skip-Thought model is trained minimising the reconstruction of the previous and the next sentence given the embedding $z(i)$. This error is then back-propagated to the encoder that tries to put as much information about $x(i)$ as possible to help reduce the decoder error while computing the previous and the next sentences.

After training, the decoders are not used anymore because they were needed only to train the encoder. The encoder can now be used to generate embeddings of sentences used for the semantic similarity task. Skip-Thought decoders work well thanks to two factors:

- **Teacher Forging**: Decoders outputs sentences one word at a time, using the true words for the target sentence, shifted by one position.
- **Probability Distribution**: The prediction is a probability distribution over words that could occur at that position given the context $z(i)$ plus the sequence of words that occurred before the current position.

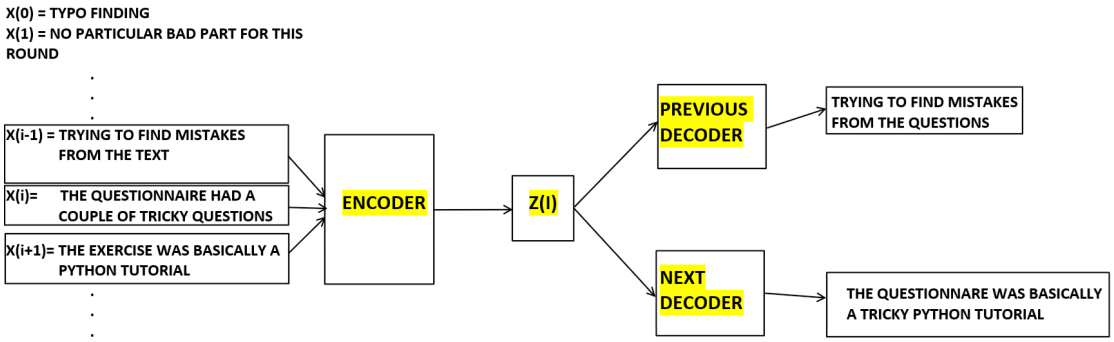

Figure 2.6. Skip Thoughts model overview.

Usually, when RNN are trained for sequence prediction, they are randomly given their true prediction label, because it helps them generalise better. In the Skip-Thought models, this does not happen and teacher forging is applied to every word because the decoder's performance does not matter since they are thrown away, and reducing the frequency of words when the teacher forging is applied will reduce accuracy.

One downside of the Skip-Thought model is that it might take weeks or even a month to train, and depending on how the university course is structured, taking action post feedback has to happen quickly.

2.1.8 The Paragram-Phrase embeddings

The dataset used for this thesis (see Table 3.1) could be considered as in-domain data, with the domain of MLBP, but this thesis' scope is to remain as general as possible, hence the interest in learning general-purpose embeddings. The Paragram-Phrase embeddings [21] are efficient and easy to use since they transfer easily across domains and while Skip-Thought vectors are better for capturing semantic affinity in terms of sentences following each other, the Paragram-Phrase embeddings try to capture paraphrastic similarity, i.e. when two sentences have the same meaning.

Paragram-Phrase embeddings encode sentences in a low-dimensional space where the cosine distance in the space corresponds to the similarity between the sentences. They are generated starting from standard word embeddings and modified using the Paraphrase Database (PPDB) [23] and the Annotated-PPDB [24] using a simple but still effective word averaging model. Paragram-Phrase embeddings are trained and tuned using only the two variations of PPDB to avoid feature engineering too much the semantic similarity task.

These embeddings were improved even further using smooth inverse frequency weighting and common component removal to avoid giving a large vector representation to common words,

using an improved version of the random walk model [25], achieving respectable results [26] in the Sentence Involving Compositional Knowledge (SICK) dataset [27] and STS (See Paragraph STS-B) as shown in Table 2.1.

Tasks	*PP*	*PP* (proj.)	*RNN*	*LSTM* (no o.g.)	*LSTM* (o.g.)	*GloVe + WR*	*PSL + WR*
STS' 12	58.7	**60.0**	48.1	51.0	46.4	56.2	59.5
STS' 13	55.8	56.8	44.7	45.2	41.5	56.6	**61.8**
STS' 14	70.9	71.3	57.7	59.8	51.5	68.5	**73.5**
STS' 15	75.8	74.8	57.2	63.9	56.0	71.7	**76.3**
SICK' 14	71.6	71.6	61.2	63.9	59.0	72.2	**72.9**

Table 2.1. Pearson's $r \times 100$ results on textual similarity tasks. Results are collected from [26] and they show the improvement done in [26] in the last two columns compared to the [21] in the first two columns. The highest scores in each row are marked in **bold**. LSTM results are divided between no output gate (no o.g.) and with output gate (o.g.). The W in WR stands for the smooth inverse frequency weighting scheme, and the R stands for removing the common components.

Results depicted in Table 2.1 are auspicious, but when this algorithm was applied to the thesis' dataset, accuracy was far from acceptable as we can see in Table 2.2. Feedback "Using Jupyter" and "Typo Finding" are not semantically close, but they achieved a score that almost represents a coincident meaning probably because they share the same number of words.

Feedback 1	*Feedback 2*	*Score*
Using Jupyter	Typo Finding	0.9974
I did not find anything to complain really.	Trying to find mistakes from the text...	0.5523
The questionnary had a couple of trick questions. Especially the one about invertible matrices [...]	The exercise were basically a python tutorial instead of actual machine learning [...]	0.7705
The course introduced the machine learning concepts in a more complicated way than necessary [...]	No particular bad part for this round.	0.7297

Table 2.2. The score represent how similar the Feedback 1 and Feedback 2 are, going from -1 (opposite meaning) to 1 (similar meaning). Refer to the chapter 3.1 to understand how the dataset is structured. Some feedback were truncated for showing purpose only.

Thus, Paragram-Phrase Embeddings were not further tested in the chapter Results.

2.1.9 Transformers

RNN and LSTM neural networks achieved excellent results in modelling natural languages, but they both have a significant drawback due to their architecture. They are sequential, and this means that parallelization during training is very limited. Parallelization is critical when there are longer sentences, because the total available RAM is often not enough to fit the entire training process, hence the training data needs to be split into multiple smaller batches. There has been some significant progress in computational efficiency and model performance using factorization [28] and conditional computation [29] but the sequential computation constraint still remains. The Transformer architecture [30] that relies only on the attention mechanism [31] allows more parallelization and can achieve promising results with less training time (if trained with multiple GPUs).

The attention is a technique that improved the precision of machine translation systems, and allows models to focus on the relevant parts of the input when needed. This technique was born to solve the problem of the encoder-decoder architecture that is incapable of remembering longer sequences, because they have to be translated into a single vector, hence becoming the

bottleneck of performance. Generally in the encoder-decoder architecture, all the intermediate states are discarded and only its final states are kept. The main idea of the attention mechanism is to use those intermediate states in order to construct the context vectors that are required by the decoders to generate the output sentence. These intermediate states are used to focus the attention on the parts of the input that are relevant for this decoding step, therefore that is why the technique has this name.

The Transformer architectures use self-attention layers, for both the encoder and the decoder as shown in Figure 2.7. The encoder is composed of six identical layers stacked on top of each other, where each layer has two sub-layers. The first is a multi-head self-attention mechanism and the second is a fully connected feed-forward network. The decoder is also composed of six identical layers, but in addition to the first two sub-layers that are also present in the encoder, the decoder has a third sub-layer that performs multi-head attention on the encoder's output. The decoder's output embeddings are shifted right to ensure that predictions for position i can depend only on the known outputs at positions that are less than i.

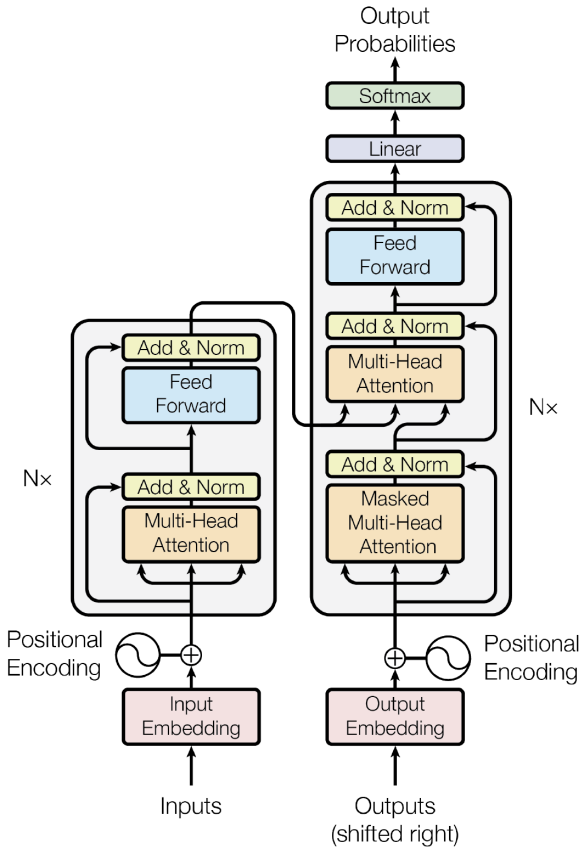

Figure 2.7. Transformer model overview. (Figure taken from paper [30])

Transformers use the attention mechanism in parallel, and this method is called "Scaled Dot-Product Attention". This mechanism allows the transformer to compute the attention function on a set of queries simultaneously using the softmax function and matrix multiplications. Multi-head attention allows the model to receive information from different representations at different positions at the same time using $h = 8$ attention layers.

The Transformer model uses multi-head attention in three different parts:

- In the encoder-decoder attention layers, when the input comes from the previous decoder layer. This allows every position in the decoder to attend over all positions in the input sequence.

- In the self-attention layers in the encoder. Each position can attend to all the position in the previous layer of the encoder.
- In the self-attention layers in the decoder. The input values of the softmax are masked out if they correspond to connections that are not permitted.

Since transformers do not use nor recurrence nor convolution, in order to use the order of the sequence it has to encode the position's information. This is done adding the positional encodings to the input embeddings at the bottom of the encoder and decoder. The self-attention mechanism enables the Transformer to learn long-term dependencies in the network. As explained before the vanishing gradient problem of RNNs [17] is more present when there are longer sequences, and to improve the Transformer performance in such sentences, the self-attention can be restricted to only a neighbour of size r in the input sequence.

The Transformer architecture shows considerable results in various tasks such as machine translation [30], document generation [32] and syntactic parsing [33] and it is included today in many state-of-the-art NLP models.

2.1.10 ELMo

Two identical words can change meaning depending on the context. This is known as "polysemy" and has been an issue that previous NLP models could not tackle, because they assumed that every word could have only one representation. Embeddings from Language Models (ELMo) [34] attempts to solve this problem by using the context when creating the embeddings, and this not only solves the problem of polysemy, but improves the overall accuracy because any word can have different meaning depending on how they are used.

There are two common strategies in NLP to utilize pre-trained language representations, one *feature-based* oriented and one that uses *fine-tuning*. ELMo uses a feature-based approach, using task-specific architectures that add pre-trained representations as additional features. ELMo architecture uses multi-layer bi-directional LSTMs trained with reversed sentences, that the authors call "backward" language model. ELMo is trained in an unsupervised and task-independent way, on a billion-word benchmark [35]. The weights are learned for each task and normalized using the softmax function. One of the major benefits of using ELMo is that it can be used in concatenation with any model with low effort, barely applying any changes to it. Unlike previous attempts for learning contextualized word vectors [36] [37], ELMo representations are more rich, extracting the hidden states of each LSTM layer for the input sequence of words, and it computes the weighted sum to obtain the embeddings of each word.

Training ELMo is not easy and it is computationally expensive like training any other language model. To combat overfitting and reduce the memory overhead, it is better to connect the weights for the forward and backward language models together [38]. If the dataset that is used for evaluation is not general-purpose but in-domain specific, it is advised to fine-tune the model on the domain-specific data to improve accuracy. Since ELMo combines the activations of different layers of LSTM, the outputs are normalized [39] and dropout is used to avoid overfitting. Adding ELMo to a model increases the sample efficiency, reducing the percentage of the training set needed to reach the same results in terms of accuracy, as shown in Figure 2.8.

2.1.11 ULMFiT

Universal Language Model Fine-tuning (ULMFiT) [41] is a transfer learning method that can be applied to any task in NLP, that is highly influenced by the progress made in the computer vision field. ULMFiT, instead of using a feature-based approach like ELMo, uses a fine-tuning oriented method that still involves a language model. This approach introduces minimal or none task-specific parameters and trains the language model by fine-tuning all the parameters for the downstream task. Before ULMFiT, deep learning models achieved state-of-the-art results on many NLP tasks, but they are trained on large corpora and they take many hours if not days to converge. Instead of focusing on *transductive* transfer [42], the authors of ULMFiT decided to

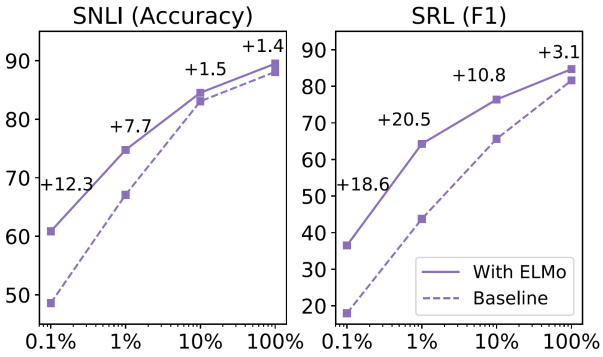

Figure 2.8. ELMo performance versus baseline for Stanford Natural Language Inference (SNLI) Corpus [40] and Semantic Role Labeling (SRL) tasks, as the training set size varies from 0.1% to 100%. (Figure taken from paper [34])

focus on a *inductive* transfer technique introduced in the word2vec model [10], still used today in most state-of-the-art NLP architectures. Inductive transfer via fine-tuning [43] [44] before ULMFiT required millions of in-domain documents to achieve good performance, which lead to overfitting on small dataset hence limiting its applicability. In NLP, novel methods have been proposed that use transfer learning on word embeddings, pre-training them to capture additional context using other tasks. This method is known as hypercolumns and it is used in ELMo [34], but it requires custom-engineered architectures, while ULMFiT proposes the same architecture across multiple tasks.

ULMFiT consists of three steps as shown in Figure 2.9, LM pre-training using general-purpose data, LM fine-tuning, and classifier fine-tuning on the specific task. Pre-training is very beneficial for tasks with small datasets, and it is done using the Wikitext-103 corpus [45], consisting of pre-processed Wikipedia articles with a total of 103 million words. According to the authors of ULMFIT, the Wikitext-103 corpus is big enough to capture the general properties of the English language. It is the most expensive part in terms of computational time, but it has to be done once and it is worth it due to the non-negligible increase in performance.

Even if the Wikitext-103 corpus is big enough, the data of the target task will most likely come from a different distribution, hence fine-tuning the LM on the data of the target task is needed. For the LM fine-tuning, two techniques are being used in the model:

- **Discriminative fine-tuning**: Since different layers capture *different types of information* [46], they should be fine-tuned to different extents. Instead of using the same learning rate for all the layers of the model, discriminative fine-tuning tunes each layer with different learning rates. It begins by fine-tuning the last layer in order to choose its learning rate, and afterwards, it computes the learning rates of the lower layers.

- **Slanted triangular learning rates**: It is another technique modified from [47] to change the learning rates, which allows it to linearly increase in the beginning, and linearly decrease the higher the iterations.

Lastly, the classifier is fine-tuned with two steps, where each step uses batch normalization [48] and dropout, with ReLU and softmax activations that output probability distributions over specific target classes of the last layer. The two steps consist of:

- **Concat pooling**: In order to avoid information loss when only the last hidden state is considered, it takes as input the last hidden layers states, as many states as they fit in the GPU memory. The fine-tuning of the classifier is the most crucial part, where aggressive fine-tuning can lead to forgetting the information captured from the language model, and shallow fine-tuning can lead to slow convergence.

- **Gradual unfreezing**: Instead of fine-tuning all the layers at once, it starts from the last layer because it contains the least general knowledge [46], and gradually unfreeze one layer and fine-tunes on the unfrozen layers until convergence. This technique is similar to the "chain-thaw" technique [49] but instead of training only one layer at a time, one layer is added at the set of thawed layers at every step.

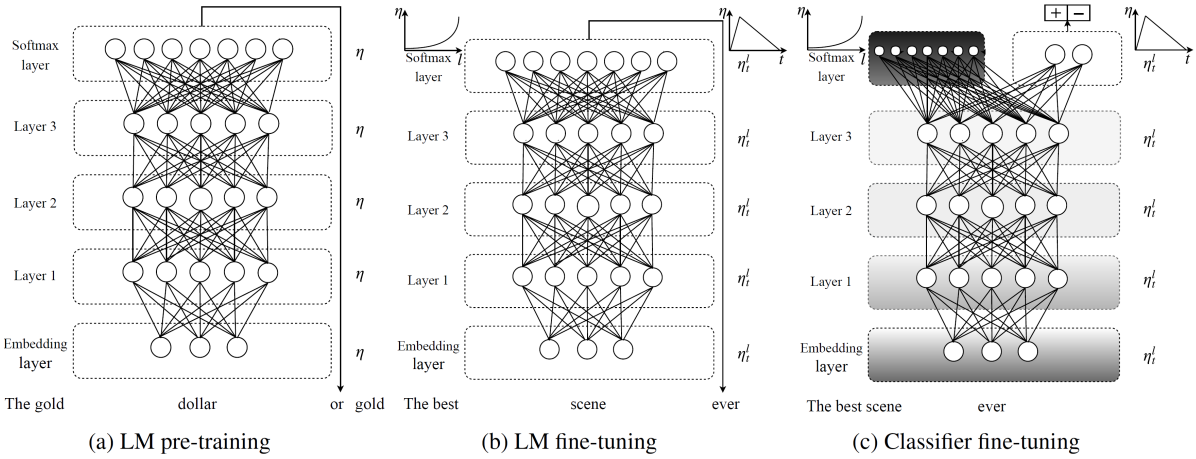

Figure 2.9. ULMFit model overview. ULMFit consists of three stages: a) General-domain LM pretraining b) Target task LM fine-tuning c) Target task classifier fine-tuning with shaded layers as unfreezing stages, and black layer that symbolize being frozen. (Figure taken from paper [41])

These techniques introduced for fine-tuning, discriminative fine-tuning, slanted triangular learning rates, concat pooling and gradual unfreezing, are useful on their own but they also improve each other making ULMFiT more general, obtaining excellent performance on a broad selection of datasets. To handle large datasets, ULMFiT divides documents into fixed-length batches. At the beginning of each batch, the model is initialized with the final state of the previous batch, and hidden states are used for mean and max-pooling. Gradients are backpropagated to the batches that were used for the final prediction. This technique is very similar to variable-length backpropagation sequences [50].

The main advantages of ULMFiT are that it is universal, it works on across different tasks with different number of documents, length and label type. It uses one architecture and training process, it is trained as a bidirectional model with a forward and backward LM, and it does not require additional in-domain data.

2.1.12 OpenAI GPT

Generative Pre-trained Transformer (GPT) [51] uses the same approach as ULMFiT, pre-training a language model on a vast general-purpose corpus, modifying the model accordingly to what task is needed, and then fine-tuning using the user-defined dataset. The advantage of this approach is that it has to learn few parameters using unlabeled data, but unlike ULMFiT, GPT uses unidirectional language modelling. GPT uses a semi-supervised approach for understanding the language model and a combination of unsupervised pre-training with a supervised fine-tuning, and like ULMFiT aims at creating a universal representation that works across different tasks.

The main difference in architecture that separates GPT from ULMFiT is that the latter uses a RNN architecture, while GPT uses a transformer model that can handle long-term dependencies in text as explained in the paragraph about Transformers. In order to provide this *task agnostic* model, GPT utilize task specific input adaptation [52] which enable GPT to process text inputs as a single continous sequence of tokens, as shown in Figure 2.10.

The GPT model is trained using the BooksCorpus dataset [53] that consists of over 7,000 different books from different genres. The GPT architecture consists of 12 layers of decoder-only

transformer with masked self-attention heads. The Adam optimizer is utilised [54] and layer normalization is used in combination with feed-forward networks.

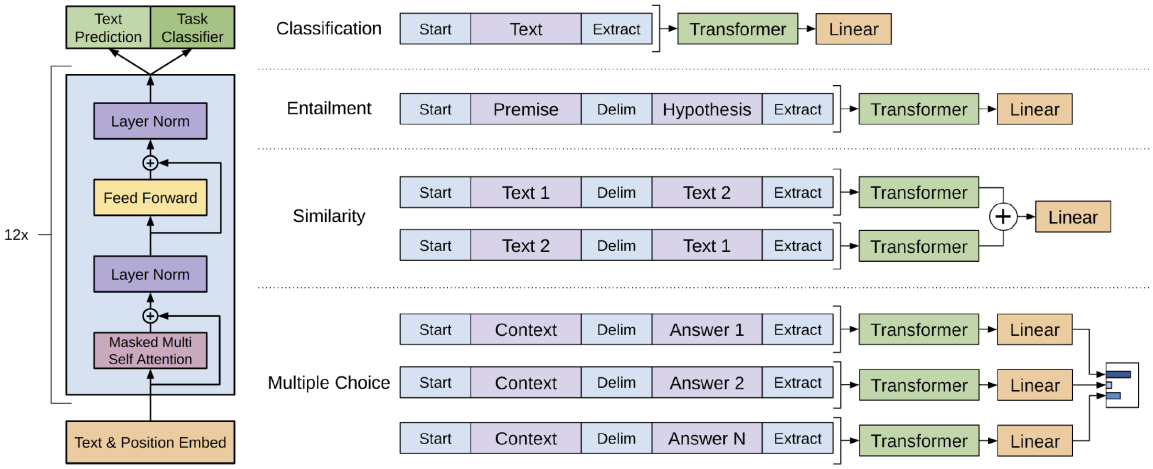

Figure 2.10. On the left there is the transformer architecture utilized by GPT. On the right, how the input is processed depending on the NLP task the algorithm is trained for. (Figure taken from paper [51])

One disadvantage of ULMFiT is that even if it is meant to be universal, it still requires heavy fine-tuning and tweaking that depends on the task it has to be applied for. GPT requires minimal extra work on fine-tuning and gives state-of-the-art or near state-of-the-art results in a wide variety of NLP tasks. The main downside is that it requires a long time for pre-training the model, even one month with 8 GPUs, but there is no reason on pre-training your model on your own if you use a natural language where the pre-trained model is available.

The authors of the paper [51] released a new version of the model called GPT-2 which is configured to use progressively a more significant number of parameters with a larger dataset obtained by scraping text on the web. Since the performance of transformers improves log-linearly with the number of parameters, even the largest model with a total of 1.5 Billions of parameters (10 times more than the original GPT) still underfits the 40GB dataset. Due to concerns about malicious use, like synthetic text generation for extremist propaganda, generating fake news or phishing purposes, only the small model was released with a size comparable to the original GPT architecture. After nine months of lockdown of the largest model, GPT-2 has been entirely released because the authors believe the model is not yet accurate enough to fool mechanisms that are designed to prove if a piece of text is generated by a human or machine.

2.1.13 BERT

Bidirectional Encoder Representations from Transformers (BERT) [55] applied the bidirectional training of Transformers, to language modelling. ELMo, ULMFiT and now BERT focussed their effort in language modelling, because LMs capture many aspects of the language needed for the NLP tasks, such as long-term dependencies [56], hierarchical relations [57], and sentiment analysis [58]. LM provides data in enough quantities for most languages and applications, data that can be easily adapted to the needed task.

The release of BERT was seen as the beginning of a new era in NLP, thanks to the clever ideas that were accumulated throught the years, that are part of Semi-supervised Sequence Learning [44], ULMFiT [41], ELMo [34], OpenAI GPT [51] and Transformers [31].

The paper proposes two model sizes, one called BERT Base that has a comparable size with the OpenAI GPT Transformer and another one called BERT Large which is a much bigger model that achieved the state-of-the-art results in the paper. BERT uses a fine-tuning approach like ULMFiT, but a feature-based approach like ELMo has its advantages. For example, not all tasks can be

easily represented with the BERT architecture and need a task-specific architecture to be added on top of the model. A feature-based approach can save computational time when an expensive representation is pre-computed once, and experiments are done on smaller versions of the model. With some modifications, BERT can be transformed into a feature-based only architecture, and it still achieves significant results, meaning that it is effective for both approaches.

Standard language models are unidirectional and they have definite limitations in capturing the semantic meanings of words and sentences of a natural language, by either scanning a text from right-to-left or left-to-right. Another disadvantage in using unidirectional models are the limitations of choices for architectures that can be used during pre-training. BERT and ULMFiT both use bidirectional architectures that use both directions at the same time, allowing the model a deeper understanding of the language context. In order to use both directions, fusing the left and right context, BERT uses a Masked Language Model (MLM) pre-training objective, inspired by the Cloze Task [59]. MLM randomly masks some of the tokens that are generated from the input with the objective to predict the vocabulary id of the masked word based on its context. In addition to the MLM technique, BERT uses a next sentence prediction (NSP) task to train text-pair representations.

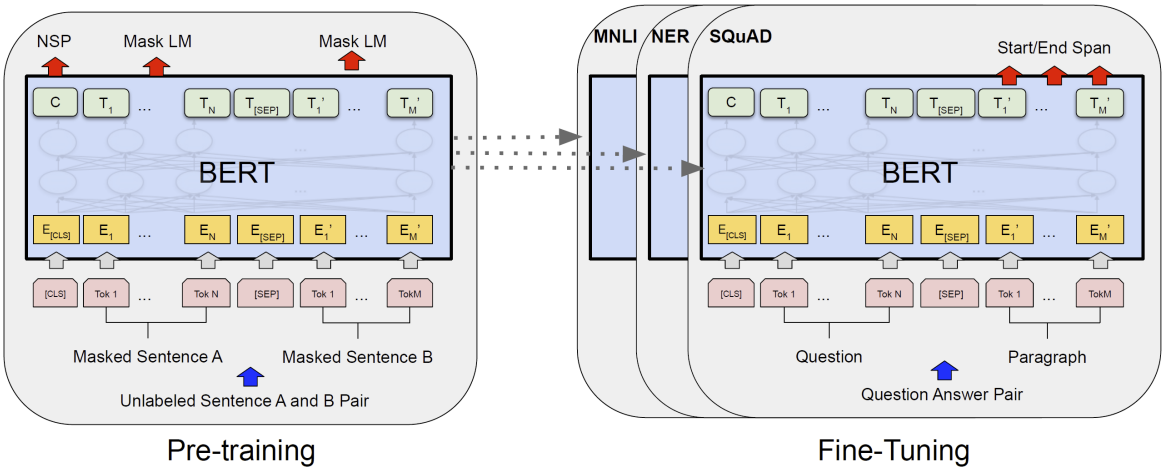

Figure 2.11. BERT architecture that consists of pre-training and fine-Tuning. Other than the output layers they share the same architecture, and the same pre-trained model is used for all the different down-stream tasks. All the parameters are fine-tuned. [CLS] and [SEP] are special symbols, the first one is added in front of every input sample, and the other is used as a separator, for example when separating questions from answers. (Figure taken from paper [55])

The BERT architecture is composed of two steps that are the unsupervised pre-training and the supervised fine-tuning. Pre-training consists of training the model on unlabeled data over different tasks. Fine-tuning consists of the model being initialized on the parameters obtained from the step before, and these parameters are then fine-tuned using labelled data from the chosen task. Each task will produce a different fine-tuned model, even if they all start from the same pre-trained parameters. One advantage of BERT is that there is minimal variation in terms of architecture between the Pre-Training step and the Fine-Tuning step, as shown in Figure 2.11.

BERT's model uses a multi-layer Transformer encoder. For the BERT base it uses L=12 layers, H=768 hidden size and A=12 attention heads with a total of 110M parameters. For BERT Large it uses L=24 layers, H=1024 hidden size and A=16 attention heads with a total of 340M parameters.

The input is represented with single sentences or sentence pairs (for example [question, answer]), meaning that for BERT a sentence is not necessarily interpreted as a linguistic sentence, but more like contiguous text. The sentences are converted into tokens using WordPiece embeddings [60] that are made to deal effectively with rare words, by using sub-words units known as wordpieces. WordPiece embeddings give a good balance between the flexibility of using single characters as embeddings (useful for languages that use ideograms instead of words, like Japanese

[61]), and the efficiency of using the entire word for decoding, without needing a system for handling unknown words. The first token of every sentence is always the special token [CLS] and the final hidden state of this token is used in classification tasks. Sentences are zipped together into one single input stream, where each sentence is separated by the special token [SEP] and in the case of a sentence pair, a learned embedding A is added on top of each token of the first sentence, and a different learned embedding B is added on top of each token of the second sentence. Usually, transformers do not encode the position of their inputs, but the same words in different places in the sentence may carry a different meaning. To overcome this problem on top of the BERT input embeddings it is added a position embedding, as shown in Figure 2.12.

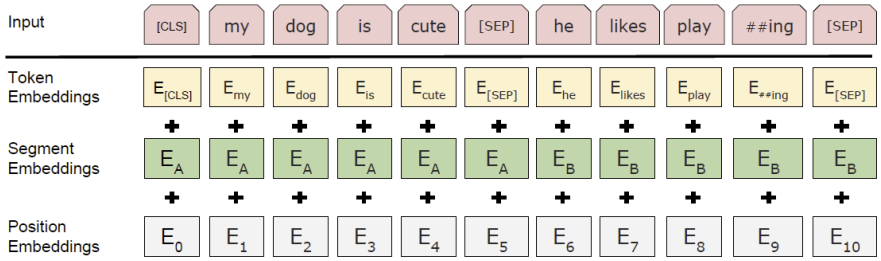

Figure 2.12. Example showing how the input embeddings for each token are generated. [CLS] token is added at the beginning of each input stream, and if the input consists of a sentence pair, a [SEP] token is added in between the sentences and after the last one. For each token the embedding is generating summing the token embedding, the segment embedding and the position embedding. (Figure taken from paper [55])

Standard language models are trained in only one direction, because if trained in a bidirectional way each word could see each other and the model will always predict the word with 100% accuracy, without learning. In order to avoid this problem, some words are randomly masked, usually the 15% of the entire input, and the model tries to predict them. This type of task of trying to learn a clean input from a corrupted version is called denoising and has been explored as a training method for encoders [62]. A downside of this method is that the pre-training step and fine-tuning step will become more different from each other, because the token [MASK] is not present in the fine-tuning step. To mitigate this problem, if a word is selected to be masked, only 80% of the times it will be replaced by the [MASK] token, while the other 10% will be replaced by a random word and it will remain unchanged for the additional 10%. The newest release of the BERT model, improved the task of masking with a new technique called Whole Word Masking. The previous masking was too easy for words that had been split into multiple wordpieces, therefore the new technique fixes this problem by masking all the wordpieces that belonged to one word before pre-processing, increasing even more the accuracy of the model in NLP tasks.

For some NLP tasks like Question Answering (QA) and Natural Language Inference (NLI) it is required a deep understanding of the relationship between the two sentences, but language models do not automatically capture this. In order to let the language model capture this relationship BERT pre-trains the model with next sentence prediction. Next sentence prediction consists into predicting if the second sentence follows the first sentence in the original input, where 50% of the time the second sentence is selected randomly, and 50% of the time the second sentence actually follows the first one. This pre-training even if it is very simple, allows BERT to have significant improvement in the QA and NLI tasks and it is done using BooksCorpus [53] and scraped English Wikipedia. These two corpora are best suited in the generation of long contiguous sequences because they are document-level corpora, instead of other sentence-level corpus such as the One Billion Word Benchmark [35].

The next step after pre-training is fine-tuning for the specific task. Fine-tuning is done using the self-attention mechanism but instead of encoding text pairs before applying the bidirectional cross attention, BERT unifies these two steps into one. Given the input task the fine-tuning process will select the correct output layer and use the [CLS] token if needed. Fine-tuning compared to pre-training is less expensive, and can be done in a few hours using a GPU.

Chapter 3

The Machine Learning Problem

3.1 Dataset

The Dataset given for this research experiment, is the collection of feedback obtained with the 2018 edition of the Aalto University course "CS-E3210 Machine Learning: Basic Principles (MLBP)"[63]. The course is divided into six rounds, and after each round students were asked to answer a feedback questionnaire, with the questions "What was the worst part of the round?" and "What was the best part of the round?" at the end of other structured questions. MLBP was one of the most enrolled course of that year, with over 700 students coming from very different backgrounds, since Aalto University has an increasing number of ERASMUS+ students, which in the Computer Science track consisted of the majority of students.

Rounds	*Number of feedback*
Introduction 1	700
Regression 2	648
Classification 3	624
Validation 4	634
Clustering 5	615
Feature Learning 6	619

Table 3.1. Number of feedback per round.

In the table 3.1 we can see the number of feedback per round, named depending on the round's topic. We can immediately see that the number of feedback decreases the more the course progressed, because some students decided to drop out after the first weeks. In the Results chapter, the rounds will be abbreviated to Intro1, Regr2, Class3, Valid4, Clust5 and Featlear6.

3.2 Pre-processing

One of the very first steps in the NLP pipeline is pre-processing the data. This step is very crucial and it usually has a significant impact on the overall accuracy of the NLP model. For Python, many NLP libraries handle this critical step, such as NLTK, Gensim, SpaCy, and NLP toolkit from scikit-learn. For this thesis NLTK [64] was used because even if it is not the fastest framework, it is the most well known and full NLP library, it has many third party extensions, and there are plenty of approaches for each NLP task.

The main steps in the pre-processing of the data are *Tokenization*, *Stemming*, *Lemmatization* and *Stopwords removal*.

3.2.1 Tokenization

Given a text, tokenization is the task of chopping the text into basic units, called tokens. As we can see from the snippet of code 3.1, tokens should not be confused with simple words, they are *characters grouped together as a useful semantic unit for processing* [65]. Tokenization based on whitespace is inadequate for many NLP applications but the NLTK library has one off-the-shelf tokenizer that implements a better method for tokenizing.

```
>>> import nltk
>>> text = "The Earth is dying, it's our responsibility."
>>> tokens = nltk.word_tokenize(text)
>>> print(tokens)
['The', 'Earth', 'is', 'dying', ',', 'it', "'s", 'our', 'responsibility', '.']
```

Figure 3.1. Snippet of python code to tokenize a string.

3.2.2 Stemming

In the English grammar, there are cases such as plurals, possessives and conjugations, where the same word is mutated, hence many words can be connected into one root word. The aim of stemming is to use heuristics to truncate the word from any affixes to the base form. NLTK includes many off-the-shelf stemmers, depending on the application. For the scope of this thesis, the Porter stemmer is chosen.

```
>>> porter = nltk.PorterStemmer()
>>> stemmed_tokens = [porter.stem(t) for t in tokens]
>>> print(stemmed_tokens)
['the', 'earth', 'is', 'die', ',', 'it', "'s", 'our', 'respons', '.']
```

Figure 3.2. Snippet of python code that shows the Porter stemmer.

3.2.3 Lemmatization

Lemmatization is similar to stemming, but instead of crude heuristics to chop the word, it uses a vocabulary and tries to do a morphological analysis of the words, trying to return the base word known as *lemma*. Lemmatization is more computational heavy compared to Stemming, and as stated in [66] there is no significant difference in accuracy between the two, so it is generally better to prefer stemming only.

3.2.4 Stopwords removal

Some words are very frequent and they carry little meaning such as 'and', 'the', 'I' and by removing them it will largely decrease the size of the dataset. Since the negations are considered stopwords, this technique is not included in the pre-processing pipeline of the thesis, because it will change the semantic meaning of the feedback.

3.2.5 Pre-processing results

Instead of removing the stopwords, some feedback are entirely removed from the dataset when they carry little or no information, such as “none”, “nothing” and so on. Figure 3.3 shows the number of feedback before and after the pre-processing, with an average dataset reduction of 10.74%. This not only improves the accuracy of the model but it will also save computing time.

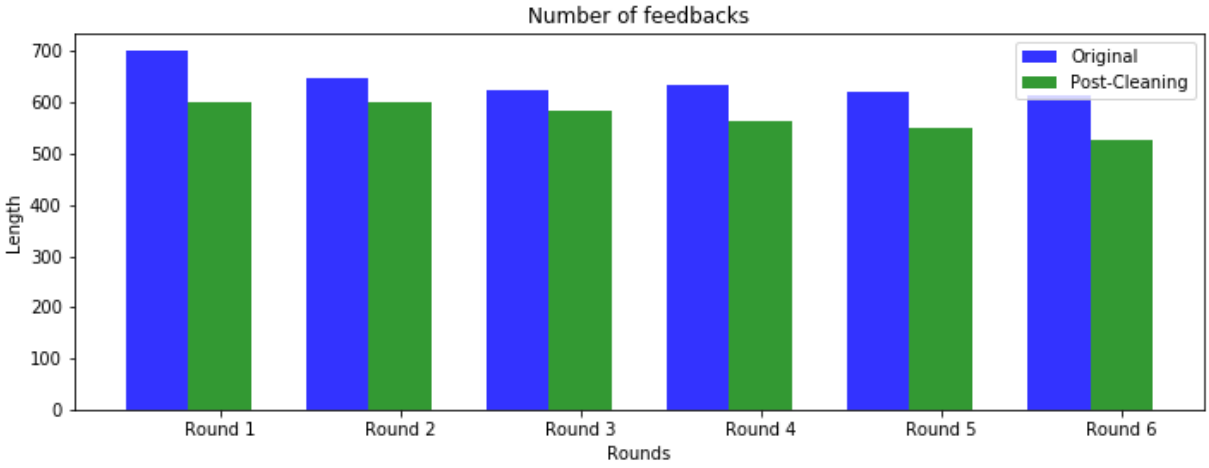

Figure 3.3. Number of feedback before and after pre-processing the data, per round.

3.3 Benchmarks

In order to assess the performance of NLP models, benchmarks and workshops were created. General Language Understanding Evaluation (GLUE) [67] is a benchmark formed by a collection of tasks in the Natural Language Understanding (NLU) field. Researchers that worked on GLUE have found that the models trained on this collection of tasks perform better than the models trained on the single tasks separately. In particular for this thesis scope the tasks Microsoft Research Paraphrase Corpus (MRPC) [68] and Semantic Textual Similarity Benchmark (STS-B) [69] are the most related. The top 10 of the leaderboard for those tasks during the year 2019, are algorithms that use Bidirectional Encoder Representations from Transformers (BERT) or variation of BERT such as Robust Optimized BERT pretraining Approach (RoBERTa) [70] or A Lite BERT (ALBERT) [71].

3.3.1 MRPC

MRPC also known as MSRP, was created due to a lack of a large-scale publicly available corpus of sentences, in a time where paraphrasing was not considered a common NLP task. MRPC consists of 5801 pairs of sentences that are naturally-occurring, non-handcrafted and it is not created from translating another language, each pair followed by a number that represents whether humans judged the two couple of sentences to be similar enough in term of semantic meaning to be a paraphrase of each other. These sentence pairs were obtained crawling the World Wide Web using heuristics to narrow the search space and to ensure some diversity between sentences.

3.3.2 STS-B

The STS-B is a benchmark created using a selection of datasets from the SemEval tasks from 2012 till 2017. SemEval stands for Semantic Evaluation and is a yearly ongoing event to discuss semantic analysis systems, evolved from the older Senseval. These events aim to explore the

meaning of the natural language and transfer these findings to computers. STS differs from both textual entailment and paraphrase detection because it captures a gradual meaning overlap instead of a binary classification based on particular relationships.

3.4 Defining a Machine Learning Problem

Every Machine Learning problem can be formally defined by three components: **data**, **hypothesis space** and **loss function** [63].

Data is composed of features and labels. Features are the properties of the data points that we use to describe the problem and are usually easy to measure. Since they describe the problem, choosing useful features is an essential first step of defining a Machine Learning problem and techniques were created to achieve excellent performance, grouped under the name of feature learning. In general, features must contain enough information to accurately describe the problem while still not being too much computationally heavy. Let $X = \{x_1, ..., x_n\}$ be the feature vector composed of n features where each $x_i \in \mathcal{X}$. $\mathcal{X}$ is the feature space which in this thesis case is equal to $\mathbb{R}$.

Labels represent a high-level property of the data, which will be the output of the algorithm. Labels are usually difficult to measure and compute, most of the times involving costly human labour, hence that is why they are the objective and final goal of the Machine Learning model. Labels are referred with the letter y and the label space is referred with the symbol $\mathcal{Y}$. The label space is usually a limited set, depending on how many ways the dataset is categorized and if the labels are continuous the task takes the name of regression, otherwise it is called classification.

The hypothesis space contains all the possible maps $h : \mathcal{X} \to \mathcal{Y}$, which map the feature space to the label space with a predicted label $\hat{y}_i = h(x_i)$ with $\hat{y}_i \simeq y_i$. The map h is also called predictor or classifier depending if it is a regression or classification task. The hypothesis space is limited by the available computational resources, reduced to a feasible subset of the original space. Knowing that the hypothesis space should be large enough to always find a map for any possible set of features as input, and small enough to be still computationally feasible, choosing how to define the hypothesis space for each Machine Learning problem is a design choice, and it depends on the data and each particular application. Another reason to keep the hypothesis space small is to avoid overfitting [72]. Overfitting happens when the map fits the data too well, understanding patterns that work only for the training data, producing an accuracy that is much higher in the training set compared to the validation set.

Given an hypothesis space, the loss function is used to measure the error between the predicted label $\hat{y}$ and the true label y. Common loss functions are the squared error loss $\mathcal{L} = (y - h(x))^2$ or the logistic loss $\mathcal{L} = log(1 + exp(-yh(x)))$.

Machine Learning problems can be divided into three classes: **supervised**, **unsupervised** and **semi-supervised**. Unsupervised algorithms do not require labelled data and can extract information using only the features. An example of unsupervised algorithms are the clustering methods that are explained in the next paragraphs. Supervised algorithms require labelled data points during training and thanks to these labels can learn to recognize better patterns in the data, allowing the algorithm to make predictions of labels of unseen data using only the features once trained. Semi-supervised is a class of machine learning algorithms that is in between Supervised and Unsupervised. Semi-supervised algorithms require a subset of the true labels during training, usually a percentage below the 30% of the entire dataset. By using only a partially labelled dataset the accuracy of the model can increase, without drastically increasing the cost of acquiring all the labels, which are often not provided.

3.5 Cluster Analysis

Cluster Analysis or clustering is the task of grouping together unl data points into clusters where points being in a cluster are more similar to each other compared to the rest of the data points.

Two types of clustering methods can be defined, hard-clustering and soft-clustering algorithms. Hard-clustering algorithms define only one possible cluster for each data point in the dataset, while soft-clustering methods allow data points to be in two or more different clusters at the same time, hence allowing clusters to overlap. Since feedback are written in natural language and due to its non-deterministic nature, feedback do not necessarily fit into one cluster but can fit more than one cluster at the same time, soft clustering might seem like the right choice of type of clustering to use. This does not often happen in the dataset of the thesis, probably because the feedback questionnaire was done immediately after the quiz section, hence for this reason and to better compare the lnLasso algorithm, only hard-clustering algorithms are considered.

There are a wide range of clustering methods and there is no universal best algorithm, because it depends on how the dataset is distributed, hence the lnLasso algorithm will be compared to a range of common clustering techniques, all already implemented in the scikit-learn [73] python library.

Clustering methods alone do not always guarantee solid results because they are heavily influenced by how the features were extracted in the first place. If the Feature extraction part can correctly model the The Machine Learning problem, even a "bad" clustering algorithm can find how the data can be clustered together, leading to good results. Cluster Analysis is usually done in a process of trial and error and it is good to remember that while sometimes the dataset cannot be clustered at all, the algorithms always try to find a solution.

3.5.1 K-Means

K-means is a a straightforward and famous technique for clustering the dataset into "k" clusters. For k-means each cluster is defined by its centre that is obtained by its mean. In the original implementation of k-means, the initial centroids are selected randomly which in the worst-case scenario can lead to slow convergence. The default implementation of k-means in the scikit-learn library uses another centroids initialization algorithm known as k-means++ which spreads the clusters uniformly after the first random centroid is selected, that results in faster convergence. After the initial centroids are selected, each data point will be assigned to the closest centroid, defining the clusters. The mean of each cluster is computed, and each mean will be the new centroid for the next step of the algorithm. These steps are repeated until convergence, where the total sum of euclidian distances from each point to its cluster is minimized.

$$\sum_{i=0}^{n} \min_{\mu_j \in C} \left(\|x_i - \mu_j\|^2 \right) \tag{3.1}$$

One of the parameters of K-means is the number of cluster k, which can be decided using the elbow method. The idea of the elbow method is to compute the inertia for a range of clusters k. Inertia is defined as the sum of the squared distance from the cluster centres for each data point which is what is minimized in the equation 3.1 and can be seen as a measure on how internally coherent the clusters are organized.

By plotting the inertia, the elbow point is the number of clusters after which the inertia starts decreasing linearly, which could be seen in the example Figure 3.4 as the point k = 3.

The elbow might be not pronounced enough to be noticed in the graph, meaning that there may not be any natural groups in the data, or it could also happen that there are more than one elbow, meaning that there is more than one way to cluster the data [74].

A variation of K-means known as Mini Batch K-means uses randomly generated subsets of the input data to reduce the computation time, but it still attempts to minimize the same objective function 3.1. In contrast to other algorithms that reduce K-means iteration time, Mini Batch K-means produces results that in general are comparable to the ones generated by the standard K-means, being slightly worse [75].

K-means has some limitations and does not work well if the clusters are not of equal size or density, if the clusters are non-spherical, or if there are multiple outliers in the data since

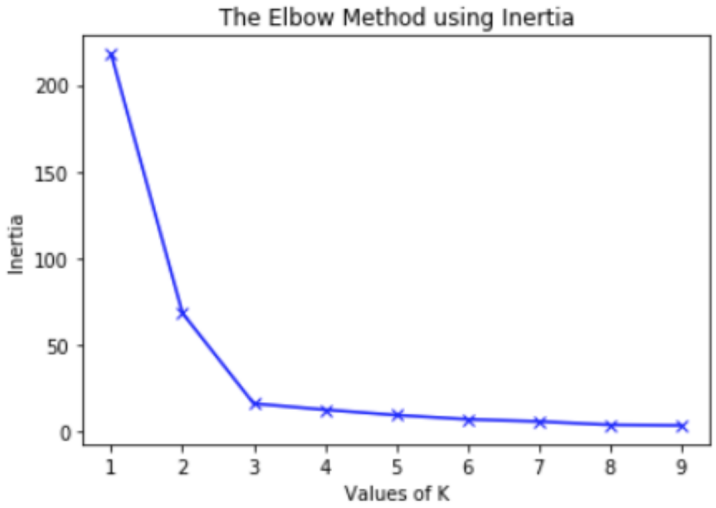

Figure 3.4. Example of inertia plotted per number of clusters k. The elbow point is found at k = 3. (Figure taken from link)

outliers have a substantial impact on the mean. Another limiting factor is that inertia is not a normalized metric, which in high dimensional space like the sentence embeddings for this thesis dataset (see Paragraph BERT) can lead to what is known as the "curse of dimensionality"; If there are too many dimensions, the Euclidian distances become flat, and every data point will seem to belong to only one cluster. This problem can be alleviated by running a dimensionality reduction algorithm such as Principal Component Analysis (PCA) before running k-means to the dataset. Running PCA before doing k-means is not the best approach, because if two points that were far in a high-dimensional space might end up close in a low-dimensional space, hence they will be clustered together even if they should not.

Due to these limitations, K-means is not the best suited algorithm to cluster educational data [76], nonetheless thanks to its popularity in other fields it is still the most used clustering algorithm in this field.

3.5.2 Affinity Propagation

Affinity propagation (AP) was first introduced in 2007 [77] and in contrast with k-means, it does not need the number of clusters as a parameter. The AP algorithm works by finding exemplars (another name for centroids) among the data points. It initially considers all data points as exemplars and with a mechanism of sending and receiving messages between data points updates the set of exemplars until convergence. The algorithm alternates two message-passing steps, to update the matrices responsibility R and availability A, both containing log probabilities.

The matrix R contains values $r(i,k)$ that measure how much x_k is suited to be an exemplar for x_i, compared to the other x_i. The matrix A contains values $a(i,k)$ that represents how much would be good for x_i to pick x_k as its exemplar, considering other points preference for x_k being their exemplar.

The messages sent in the iterations are the responsibilities that are based on the similarity function s. The similarity functions is the negative euclidian distance between points $s(i,k) = -||x_i - x_k||^2$. The responsibility messages are defined by:

$$r(i,k) \leftarrow s(i,k) - \max_{k's,t,k' \neq k} \{a\,(i,k') + s\,(i,k')\} \tag{3.2}$$

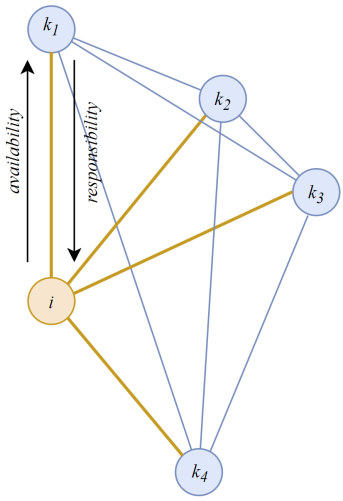

Figure 3.5. Example of graph with five data points, showing how the message are sent between data points. (Figure taken from link)

Availability is defined by the following formula:

$$a(i,k) \leftarrow \min\left\{0, r(k,k) + \sum_{i's.t.i' \notin \{i,k\}} \max\left\{0, r\left(i',k\right)\right\}\right\} \forall i \neq k. \tag{3.3}$$

$$a(k,k) \leftarrow \sum_{i' \neq k} \max\left(0, r\left(i',k\right)\right) \forall i = k. \tag{3.4}$$

The initial values for availability and responsibility are set to zero in the first iterations and updated with formulas 3.2, 3.3, and 3.4 until convergence. The AP algorithm will output for each data point which one is the most probable cluster, given by the sum of the availability matrix plus the responsibility matrix. AP can fail to converge due to oscillations, hence a damping factor λ is introduced that gives stability and faster convergence. λ is a real value between 0 and 1 and updates the responsibility and availability matrices as follows:

$$\begin{aligned} R(i,j)_t &= (1-\lambda) \cdot R(i,j)_t + \lambda \cdot R(i,j)_{t-1} \\ A(i,j)_t &= (1-\lambda) \cdot A(i,j)_t + \lambda \cdot A(i,j)_{t-1} \end{aligned} \tag{3.5}$$

In the scikit-learn library implementation, the default value of λ is 0.5, and it should be increased when the algorithm does not converge due to oscillations.

One of the most critical parameters for AP is the preference, that measures how likely points with larger values are chosen to be exemplars, hence how many points are used to define the clusters. Preference is set to the diagonal of the S matrix, and this parameter influences the number of clusters, but there is no clear relation between the two values [78]. If no preference is selected, the default value is the median of the S matrix.

AP has been succesfully used for a wide range of tasks, such as image processing [79], gene detection [80] and it has also been used for text clustering [78]. The main drawback of AP is its complexity, that is in the order of $O(N^2T)$ where N are the number of samples and T are the iterations till convergence, limiting this clustering algorithm to small to medium datasets.

3.5.3 Spectral Clustering

The goal of the Spectral Clustering (SPC) algorithm is to cluster together data points that are connected which are not necessarily compact (see Figure 3.6), overcoming some of the negative aspects of k-means if the dataset is in a space with dimension bigger than $\mathbb{R}^2$.

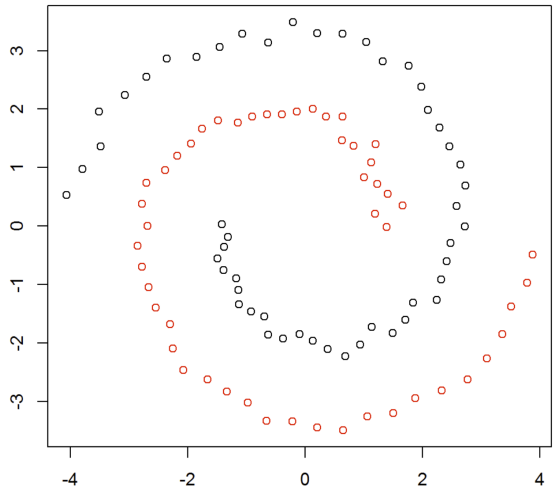

Figure 3.6. Example of dataset containing two clusters that are connected together but are not compact. (Figure taken from link)

The aim of the algorithm is constructing the affinity graph $G = (V, E)$ thanks to which the clustering problem can be formulated as a graph partitioning problem. SPC tries to find a partitioning such that the edges between different clusters have small weight and the edges within a cluster have high weight. The vertices v_i of the graph represent the data points x_i and two different vertices v_i and v_j are connected if the similarity s_{ij} is positive or more significant than a certain threshold.

The first step of the algorithm is to compute the affinity matrix A, which represents the data's structure. This can either be done by computing the graph of nearest neighbour, or via Radial Basis Function (RBF). The RBF function, also known as Gaussian Kernel is defined as follows:

$$K(x_i, x_j) = \exp\left(-\alpha \|x_i - x_j\|^2\right) \tag{3.6}$$

Where α is a free parameter that sometimes is written as $\alpha = \frac{1}{2\sigma^2}$. Using the Gaussian Kernel if two points are close the similarity between them is $s_{ij} \simeq 1$ and if two points are distant $s_{ij} \simeq 0$. Choosing the right α is still an open issue. Usually σ is set to an empiric value that is 10% or 20% of the maximum Euclidian distance between data points in the dataset [81] which makes the SPC algorithm very sensitive to outliers. Another option is to run the Spectral Clustering algorithm with different choices of σ and selecting the one that gives less distortion [82], but it might happen that the dataset has various local statistics that does not allow all the data to fit using only one value of σ. Another choice for σ is using a local scaling that is unique for every data point that depends on the distance between itself and the K-nearest neighbour [83]. This method does work well on synthetic data, but has limited success on real-world data [84].

SPC with K-nearest neighbour [85] partially removes the problems that might arise in clusters with varying density and the unreliability of selecting a σ for RBF, by replacing this parameter with the number of neighbours K which can be chosen easily. The affinity matrix is constructed using an affinity graph using the data points as vertices and similarities as edges. For each point k symmetric neighbours are found using the euclidian distance as a metric. For each couple of points x_i and x_j the edge v_{ij} is created if x_i is a neighbour of x_j and viceversa. The vertices with less than half of the average number of edges are connected with the first $k/2$ vertices and half of their neighbour if their degree (see definition slightly below) is less than $k/2$. For each point x_i and x_j their affinity a_{ij} is set to one if there is an edge v_{ij}, otherwise is set to zero, and all the elements in the diagonal of A are set to zero. This approach creates a sparse affinity matrix that captures the structure of the data.

Some algorithms use the top eigenvectors [86] [87] of the affinity matrix for clustering. The top eigenvectors contain information useful for clustering, but this does not always happen. As

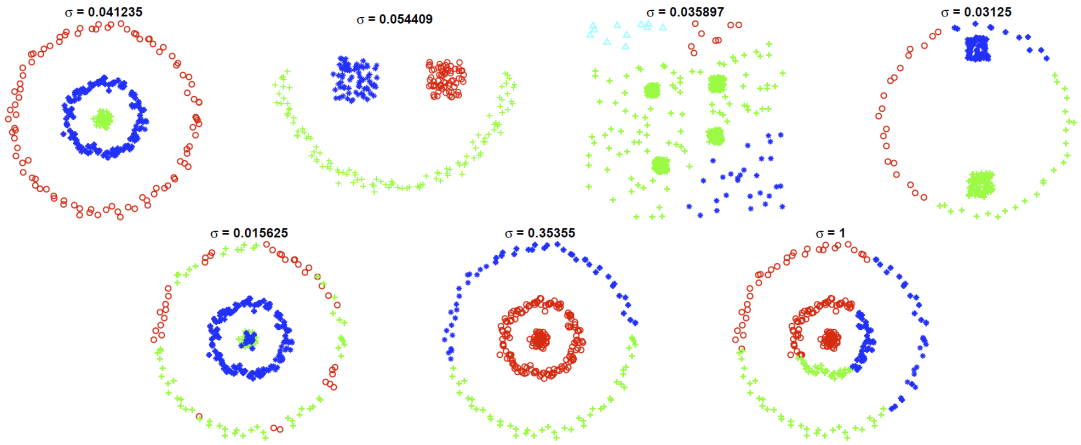

Figure 3.7. Spectral clustering with different choices of σ using RBF. The top row shows the correct choice of σ for four different datasets. The bottom row shows how the algorithm reacts at different wrong choices of σ using the first dataset. (Figure taken from paper [83])

stated in [88] when clusters are not balanced and/or cluster have different shapes the information that is contained in the eigenvectors is not enough to represent clusters, some eigenvectors can represent the same cluster and some cluster can be missed. The idea present in [88] is used by [85] with the signless Laplacian defined below.

After computing the affinity matrix A the next step in [85] is computing the diagonal degree D matrix defined as $d_i = \sum_j a_{ij}$. After that the signless Laplacian [89] $M = D + A$ is computed. Similar to [88] the top w eigenvectors of M are computed, and analyzed those who do not change sign, up to a standard deviation. The w value is equal to the double of expected clusters, meaning that each cluster can be represented by at least one eigenvector. After computing the overlapping eigenvectors that are related to the same cluster, the modularity is calculated for each overlapping eigenvector in order to choose the best eigenvector that can represent the cluster. Overlapping eigenvectors are found knowing that data points that are near the cluster's centre have small entries in other eigenvectors [85] (unless they represent the same cluster), so if an entry in an eigenvector is bigger than a small value ϵ, e.g. $\epsilon = 0.001$ the two eigenvectors overlap. Modularity [90] [91] is a metric that defines how good a division is, when splitting a network and it is defined as:

$$Q = \sum_i e_{ii} - \sum_{ijk} e_{ij} e_{ki} = \operatorname{Tr} \mathrm{e} - \left\| \mathrm{e}^2 \right\| \tag{3.7}$$

The matrix e contains the fraction of all the edges that connect different networks, where e_{ij} is the edge between network i and network j, having a total of k networks. The trace of the matrix $\operatorname{Tr} \mathbf{e} = \Sigma_i e_{ii}$ is the subset of edges in the division that connects vertices inside the same division. A good division should have a high value of trace but on its own it is not a good indicator of the quality of the division, because putting all the vertices in the same division will provide a $\operatorname{Tr} \mathbf{e} = 1$ which is the maximum value but it does not provide any information about the network. To balance the trace, it is subtracted the $\|\mathrm{e}^2\|$ which is the sum of all the elements in the matrix e. If the number of edges inside the network is not better than random, the modularity will be $Q = 0$, and values that are close to $Q = 1$ indicate a strong community structure. Usually the modularity values are in the range between 0.3 and 0.7.

After obtaining the best eigenvector that can represent the cluster, each data point is assigned to the closest cluster. SPC has some similarities with Kernel PCA [92] which has also been used for clustering. The differences lay in how the matrix A is normalized to form the matrix L. This normalization improves the performance [82]. Like k-means, SPC needs the number of cluster as input but has the advantage of detecting arbitrary shaped clusters, making it an acceptable choice for clustering educational data [66] if the number of clusters is small.

3.5.4 DBSCAN

Density-based spatial clustering of applications with noise (DBSCAN) [93] is another conventional clustering algorithm that was invented to solve the limitations of clustering extensive spatial database. In order to be applied to large datasets, the clustering algorithm should require minimal domain knowledge because it is often not provided and/or not known, it should be able to discover clusters of arbitrary shape and being efficient enough to handle large datasets. Previous density-based attempts [94] to identify clusters partitioned the data using non-overlapping cells and histograms, and even if they were able to recognize clusters of any shape, performances were not good because they had to compute multidimensional histograms.

The main idea behind DBSCAN is that humans can recognize clusters by merely looking at the distribution of data points, because clusters have a high density inside them, which is significantly higher than outside them, where outliers and noise are present. This concept is translated into an algorithm using the following definitions:

$$\mathrm{N_{Eps}(p)} = \{\mathrm{q} \in D | \text{ dist } (\mathrm{p}, \mathrm{q}) \leq \mathrm{Eps}\} \tag{3.8}$$

The Eps-Neighbourhood of a point is defined by the points that have a distance that is less or equal than Eps. A natural approach would be to require for each point in a cluster to have at least a minimum number of points in the Eps-Neighbourhood of that point. This approach does not work because there are two different types of points in a cluster. Those that are inside the cluster, which are called "core points" and those which stand in the border of the cluster, referred as "border points". For border points the Eps-Neighbourhood will contain fewer data points compared to the Eps-Neighbourhood of a core point, which will lead on setting a minimum number of points that is going to be small enough to include also noise inside the cluster, hence why this approach does not work.

A point p is *directly density-reachable* from a point q if:

1. $\mathrm{p} \in \mathrm{N_{Eps}(q)}$
2. $|\mathrm{N_{Eps}(q)}| \geq$ MinPts (core point condition)

A point p is *density-reachable* from a point q if there is a chain of points $p_1, ..., p_n, p1 = q, p_n = n$ such that p_{i+1} is directly density-reachable from p_i. Two border points can be not density-reachable from each other but there must be a core point in the cluster from which both border points are density-reachable. This is translated into the definition of *density-connected*, where two points p and q are density-connected if there is a point such that both p and q are density-reachable. Thanks to the previous definitions, the next two will define what a cluster is and what noise is, with noise being the set of points not belonging to any cluster.

A cluster C is a non-empty subset of the data points D that satisfy the following conditions:

1. $\forall p, q$: if $p \in C$ and q is density-reachable from p than $q \in C$ (Maximality)
2. $\forall p, q \in C$: p is density-connected to q (Connectivity)

And noise is defined as: $noise = \{p \in D \mid \forall i : p \notin C_i\}$.

The two DBSCAN algorithm parameters eps and min_samples which was previously referred as MinPts, define how dense must be the points to be treated as clusters instead of noise. More precisely the min_samples parameter controls how the algorithm is tolerant toward noise, while eps controls the local neighbourhood of the points and is a crucial parameter that strongly depends on the dataset and cannot be left at the default value. When eps is chosen too small, most of the data will be labelled as noise, when it is chosen too large it causes close cluster to be merged into a bigger cluster, with eventually all clusters merged into a single cluster. The authors of the paper [93] proposed a heuristic to determine the parameter eps and min_samples for the "thinnest" cluster. This heuristics works with this assumption:

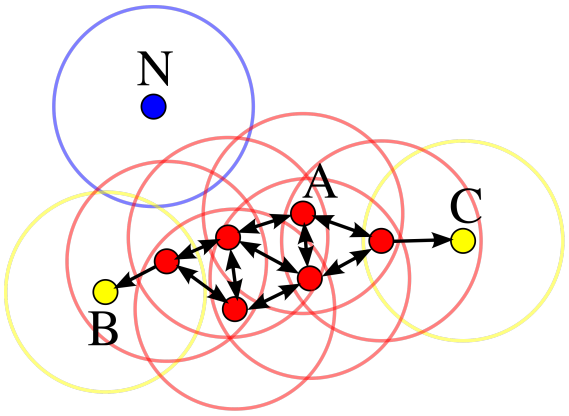

Figure 3.8. Figure showing how the points are clustered together. Point A and all the other red points are core points, because the eps radius contains at least min_samples=4 points, including the point itself. The red points are reachable from each other, hence they form a single cluster. The yellow points B and C are not core points but border points, because they are still reachable from A thus they are part of the cluster. Point N is a noise point, because it is not reachable from any cluster and it is not a core point. (Figure taken from link)

Given the distance d from a point p to its k-th nearest neighbour, the d-neighbourhood of p contains exactly $k+1$ points for almost all points p. The d-neighbourhood will contain more than $k+1$ points only if various points have the same distance d from p, which is rare. Changing the value of k does not affect d significantly. This happens only if the points of the nearest neighbourhood are located into a line which is generally not true for points in a cluster. Plotting these values of k-distances for different values of k in descending order will reveal some information about the density distribution of the dataset. Similar to the elbow method used for finding the number of cluster k in k-means, finding the point after which the k-dist start decreasing linearly can help the user in estimating the percentage of noise, that can be used as a starting value for eps.

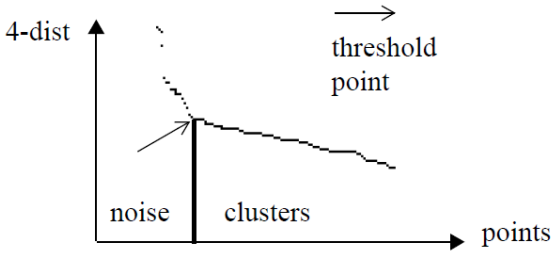

Figure 3.9. Example of graph plotting the k-distance with $k = 4$. The authors of the paper [93] noticed that the graph did not change much for values greater than 4 for their dataset, hence why they chose the $k = 4$ value. The "elbow" point is emphasized by an arrow, showing that the values before that point are noise outside the clusters, indicating that the eps value should be greater than that value. (Figure taken from paper [93])

In order to find the Nearest Neighbours, the scikit-learn implementation uses either the ball tree or kd-tree algorithm. The kd-tree is a data structure used to speed up nearest neighbour search. It works by building a tree by repeatedly splitting the dataset into halves for each level of the tree until you have m points left. Given a point x the k-nearest neighbours are found by firstly fiding in which leaf of the tree the point x is. In that leaf so in that subset of the dataset, the k points that have the minimum distance are found, and the most significant distance is used as a radius to generate a circle around point x. From that leaf, all the tree leaves are visited excluding the nodes where the circle does not cross the border of that subset, hence a neighbour with a distance that is lower than the ones that were already found could not be possibly present. If during the visit one node contains a data point with a distance that is lower than the one

previously used to compute the circle, the distance is updated and the previous data point is removed from the set of nearest-neighbours. Given the nature of the algorithm, kd-tree works better only if it is working on a low-dimension space. Increasing the dimension will reduce the effect of pruning, making the algorithm as slow as the worst-case scenario for every iteration.

The ball-tree algorithm overcomes the limits in high-dimensional space of kd-tree. Instead of dividing the dataset into boxes by repeatedly halving the dataset for each level of the tree, ball-tree algorithm halves into spheres. It selects a random point and from it, the point that is furthest away. From this point it selects the point that is furthest away again, and defines a direction given by the two points, which with high probability is the direction where the dataset is highly spread. Given this direction all the points of the dataset are projected into this line, and the median is found dividing the dataset in two. The mean is computed for the points that are on the left of the median and for the points on the right. The mean is used to define the center of two spheres with radius from the mean till the furthest point. The algorithm works the same as kd-tree from this point, by building a tree with the spheres and visiting this tree until all the neighbours are found.

The scikit-learn implementation of DBSCAN is deterministic and always generates the same cluster if the order of data points is preserved. This implementation suffers from memory consumption problems if ball trees or kd-trees cannot be used and a full similarity matrix needs to be computed, e.g. in the case of sparse matrices.

3.5.5 Summary of clustering techniques pros and cons

The lnLasso algorithm will be compared with other four hard clustering techniques and they have advantages or disadvantages depending on how the dataset is structured. Summarizing the pros and cons of each clustering technique:

- K-means does works well with clusters of the same size, if they have a flat geometry and if there are few clusters.
- Affinity propagation works with uneven clusters, even if there are too many of them and if they have non-flat geometry, but it is the slowest algorithm.
- Spectral Clustering works better with few clusters of even size, but it can detect clusters with non-flat geometry.
- DBSCAN works with non-flat geometry and uneven cluster size, but if the dataset is too big in a high dimensional space, it might run into memory issues.

In Figure 3.10 is represented how the different clustering algorithm performs in different datasets.

3.6 Clustering Accuracy

To evaluate the performance of the standard clustering techniques versus lnLasso, different accuracy metrics were used. The different metrics used can be grouped into two categories, those that rely on external indexes and those that find internal indexes. The metrics that use external indexes require external information that is not produced by the clustering algorithm, which is the correct label obtained through human labour, while metrics that use internal indexes do not need any additional information. Metrics that use internal indexes are found to be more precise [95] when capturing the clustering structure, but metrics that use external indexes can better recognize if the end-result of clustering is tuning to what the user needs. The metrics that will be used to evaluate the clustering performance are: **Confusion Matrix**, **Adjusted Rand-Index**, **Normalized Mutual Information**, **Fowlkes-Mallows score**, **V-measure**, **Silhouette Coefficient**, **Davies-Bouldin Index**, **Calinski-Harabasz Index** and **Density-Based Clustering Validation Index**.

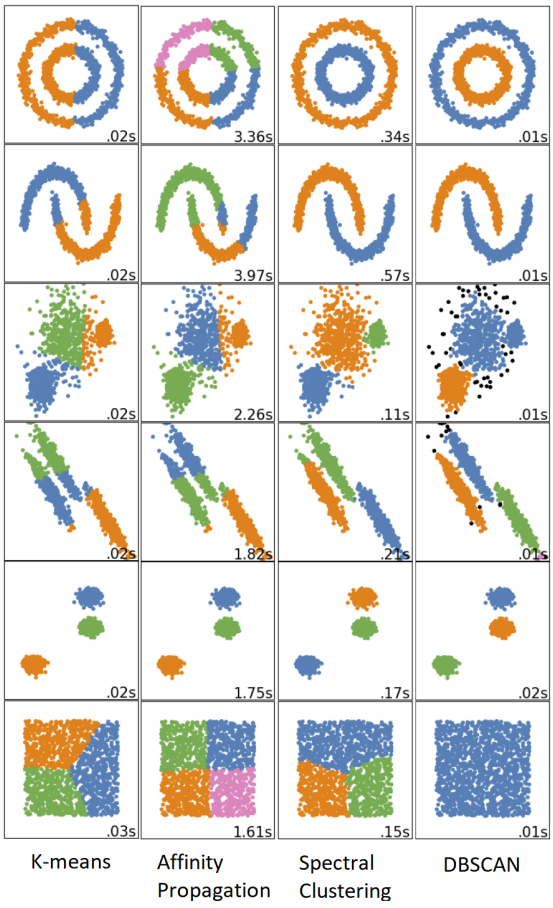

Figure 3.10. Representation on how different clustering algorithm perform on different datasets and how much time is needed to converge. (Figure taken from link)

3.6.1 Confusion Matrix

One way to construct the best accuracy metric to evaluate the clustering accuracy needs to use the expensive human labour to label each data point manually, assigning them a cluster. This process that is often referred as "the gold standard" in the literature, in order to limit the human error requires multiple humans and it is usually done with three people, where two of them initially label each data point, and if they don't agree on some data points the third person will act as a judge and decide the final label.

To compare the lnLasso accuracy in binary classification, a label was assigned for each feedback for the first three rounds of the Dataset, Introduction, Regression and Classification. This label will contain "1" if the feedback is related to the exercise session or quiz, "-1" if the feedback is related to the lecture, or to a concept that the student did not understand that is contained in the lecture's book. If a feedback does not relate to both labels, it will be assigned the label "0" and it will be treated as noise.

An approach to evaluate clustering performance can be to simply divide the number of predicted labels that were correct with the total number of feedback. When a feedback has been labelled as "0" it is excluded from the total number of feedbacks, because all the clustering algorithms presented before will try to cluster these outliers, except for DBSCAN. This approach is not very well suited if the two classes are unbalanced, which is the case of the dataset of the thesis.

In order to better evaluate the accuracy, a confusion matrix is needed. A confusion matrix is a tool that is often used in Machine Learning to summarize the results of the prediction in

Label	*Feedback*
0	there is no worst part here. every exercise complements each other well.
-1	not enough examples during the lectures!
1	implementing the gradient and empirical risk. matrix-operations are painful with python.

Table 3.2. Example of feedback with their corresponding assigned label.

a classification problem. The matrix is constructed showing the predicted labels as rows of the matrix and the actual labels as columns of the matrix.

Given the binary label $y_i = \{-1,1\}$ a True Positive (TP) is defined as a data point that should have label "1" is correctly classified with label "1". A True Negative (TN) is defined as a data point that should have label "-1" is correctly classified with the label "-1". A False Positive (FP) is defined as a data point that should have label "-1", but the predicted label is "1". A False Negative (FN) is defined as a data point that should have label "1", but the predicted label is "-1". Given these definitions the accuracy is defined as [96]:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.9}$$

This accuracy is also referred as Rand-Index (RI) and has a value that goes between zero and the absolute value one.

3.6.2 Adjusted Rand-Index

The RI defined above does not take into account that random clustering should have a score that is a constant value of zero. This problem of RI is more evident when there are many different clusters. In case of a number of clusters that is greater than 2, TN becomes all the cases when given two data points that should be in a different cluster are in a different cluster. Given the higher number of clusters, there is a higher probability that two data points are in a separate cluster even when simply assigning a random label. This is not the case of this thesis because there are only two different clusters, but the chance normalization is still an advantage over the simple RI.

Defining the set of possible values that the true labels can be as $Y = \{Y_1, Y_2, ..., Y_s\}$ and the set of possible values that the predicted labels can be as $X = \{X_1, X_2, ..., X_s\}$ the overlap between X and Y can be illustrated using the contingency matrix where each entry n_{ij} is the number of data points in common between X_i and Y_j, $n_{ij} = |X_i \cap Y_j|$.

X/Y	Y_1	Y_2	$\dots$	Y_s	Sums
X_1	n_{11}	n_{12}	$\dots$	n_{1s}	a_1
X_2	n_{21}	n_{22}	$\dots$	n_{2s}	a_2
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
X_r	n_{r1}	n_{r2}	$\dots$	n_{rs}	a_r
Sums	b_1	b_2	$\dots$	b_s	

Note that in the binary case of only two clusters, the contingency matrix and the confusion matrix are the same matrix, with the contingency matrix having an extra sum for each row and column.

Using the contingency matrix, the Adjusted Rand-Index (ARI) [97] is defined as:

$$ARI = \frac{\sum_{ij}\binom{n_{ij}}{2} - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}}{\frac{1}{2}\left[\sum_i \binom{a_i}{2} + \sum_j \binom{b_j}{2}\right] - \left[\sum_i \binom{a_i}{2}\sum_j \binom{b_j}{2}\right] / \binom{n}{2}} \tag{3.10}$$

ARI has an expected value of zero for randomly assigned labels, it is symmetric, ignores permutations and has a value between "-1" for bad clustering and "1" for perfect clustering. No assumptions are made on the cluster structure, so it can be used to validate any clustering algorithm.

3.6.3 Normalized Mutual Information

Mutual Information (MI) measures the agreement between two vectors of labels. Two different normalized versions are defined, Normalized Mutual Information (NMI) which is more used in the literature, and the Adjusted Mutual Information (AMI) that like ARI is adjusted against chance [73].

Given the set of possible values defined for ARI X and Y in the chapter above, the entropy is defined as:

$$H(Y) = -\sum_{i=1}^{|Y|} P(i)\log(P(i)) \tag{3.11}$$

where $P(i) = |Y_i|/N$ is the probability that a data point picked randomly will have the label Y_i. The analogous entropy for X is defined as:

$$H(X) = -\sum_{j=1}^{|X|} P'(j)\log(P'(j)) \tag{3.12}$$

where $P'(j) = |X_i|/N$ is the probability that a data point picked randomly will have the label X_i. MI is defined as:

$$\mathrm{MI}(Y,X) = \sum_{i=1}^{|Y|}\sum_{j=1}^{|X|} P(i,j)\log\left(\frac{P(i,j)}{P(i)P'(j)}\right) \tag{3.13}$$

where $P(i,j) = |Y_i \cap X_i|/N$ is the probability that a randomly selected data point will have the label X_i and Y_i. The NMI is defined as:

$$\mathrm{NMI}(Y,X) = \frac{\mathrm{MI}(Y,X)}{mean(H(Y),H(X))} \tag{3.14}$$

Like RI, NMI and MI are not adjusted for chance and this problem will increase when the number of cluster increases. Defining $a_i = |Y_i|$ and $b_j = |X_i|$ that are respectively the number of elements that have the label Y_i and X_i the expected value $E[MI]$ can be computed using the following formula [98]:

$$E[\mathrm{MI}(Y,X)] = \sum_{i=1}^{|Y|}\sum_{j=1}^{|X|}\sum_{n_{ij}=(a_i+b_j-N)^+}^{\min(a_i,b_j)} \frac{n_{ij}}{N}\log\left(\frac{N\cdot n_{ij}}{a_i b_j}\right) \frac{a_i!b_j!(N-a_i)!(N-b_j)!}{N!n_{ij}!(a_i-n_{ij})!(b_j-n_{ij})!(N-a_i-b_j+n_{ij})!} \tag{3.15}$$

Using the expected value, AMI can be defined as:

$$\text{AMI} = \frac{\text{MI} - E[\text{MI}]}{\text{mean}(H(U), H(V)) - E[\text{MI}]} \tag{3.16}$$

Like ARI, AMI gives a constant number zero for randomly generated assigned labels, but has a value that goes between zero and the perfect value one. For NMI and AMI various averaging method are used depending on the field where NMI and AMI are applied. This is a parameter that can be changed in the scikit-learn [73] implementation called "average_method".

3.6.4 Fowlkes-Mallows score

The Fowlkes-Mallows score (FMI) [99] is defined as :

$$\text{FMI} = \frac{\text{TP}}{\sqrt{(\text{TP} + \text{FP})(\text{TP} + \text{FN})}} \tag{3.17}$$

The FMI ranges between zero and the perfect score one. Like AMI and ARI, FMI has a constant value of zero for randomly assigned labels

3.6.5 V-measure

Given the definition of homogeneity and completeness:

$$\begin{aligned} h &= 1 - \frac{H(C|K)}{H(C)} \\ c &= 1 - \frac{H(K|C)}{H(K)} \end{aligned} \tag{3.18}$$

where $H(C|K)$ is the conditional entropy of the labels given the cluster assignment, defined as:

$$H(C|K) = -\sum_{c=1}^{|C|}\sum_{k=1}^{|K|}\frac{n_{c,k}}{n} \cdot \log\left(\frac{n_{c,k}}{n_k}\right) \tag{3.19}$$

and $H(C)$ is the entropy defined with the formula 3.11. n is the total number of labels, n_c and n_k are the total number of labels of class c and k and $n_{c,k}$ is the total number of labels from c that are in k. Homogeneity measures how much each predicted cluster contains only data points that their true labels are in that cluster. Completeness is symmetrical to homogeneity and it measures how many data points with the same true label ended up in the same predicted cluster. Both homogeneity and completeness have a score that goes between zero and the best value one.

The definition of the entropy-based cluster evaluation metric V-Measure [100] is the harmonic mean of homogeneity and completeness:

$$\text{V} = 2 \cdot \frac{h \cdot c}{h + c} \tag{3.20}$$

V-measure defined like this is equivalent to NMI where the aggregation function is the arithmetic mean [101]. The scikit-learn [73] implementation uses this formula:

$$\text{V} = \frac{(1 + \beta) \times h \times c}{(\beta \times h + c)} \tag{3.21}$$

where β is a parameter that decides how much the V-measure is influenced by the homogeneity compared to the completeness.

Like NMI, MI and RI, V-measure does not assign a constant value zero to randomly assigned labels, but as shown in Figure 3.11 this phenomenon is very limited when the number of clusters is meager, as the case of the dataset of the thesis. V-Measure has a value that goes from the awful score zero to the absolute score one.

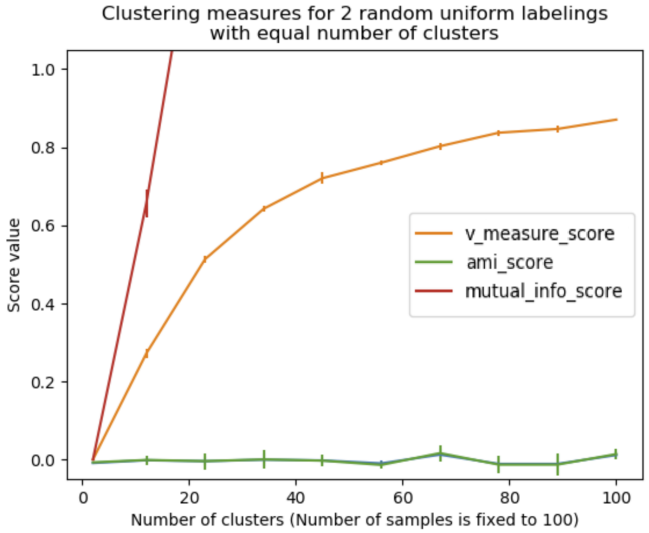

Figure 3.11. Graph showing how the different V-measure AMI and MI scores vary in the case of random labels per number of clusters. (Figure taken from link)

3.6.6 Silhouette Coefficient

All the previous clustering evaluation metrics require the knowledge of the true labels, because they are clustering performance metrics that rely on external indexes. The Silhouette Coefficient (SC) [102] is the first example of clustering metric that relies on internal indexes and they are the cluster tightness and separation.

SC is defined as:

$$\text{SC} = \frac{b - a}{max(a, b)} \tag{3.22}$$

where a is the mean distance between a data point and all the other points with the same true label, and b is the mean distance between a data point and the next nearest cluster.

SC has a value between "-1" for incorrect clustering and "1" for dense clustering. The value "0" indicates that clusters overlap. Higher scores of SC means that the clusters are dense and separate. SC can be used to select the number of clusters like the elbow method illustrated in the chapter K-means. In order to select the best number of cluster the average SC score for every data point must be computed and plotted along with the width of every cluster per number of clusters k. Every amount of cluster k with clusters having an average SC score below the total average are discarded. If two or more numbers of clusters k are left, it is selected the number of clusters that provides a more homogeneous distribution. One drawback of the SC is that it has a higher value for clustering algorithms that generate convex clusters, like K-means, compared to other clustering methods like the density-based DBSCAN.

3.6.7 Davies-Bouldin Index

The Davies-Bouldin index (DBI) [103] is another evaluation metric that uses internal indexes to assess clustering performance. Given the definition of similarity as:

$$\mathrm{R_{ij}} = \frac{s_i + s_j}{d_{ij}} \tag{3.23}$$

where s_i is the average distance between each data point of cluster i and the centroid of that cluster and d_{ij} is the distance between the cluster centroids i and j.

The DBI is defined as:

$$\mathrm{DBI} = \frac{1}{k}\sum_{i=1}^{k} \max_{i \neq j} R_{ij} \tag{3.24}$$

where k is the number of clusters. Computing the DBI is less computational expensive compared to the SC, and lower values of DBI mean a better clustering with zero as the best possible value. Like SC, DBI gives higher scores for convex clustering algorithm such as K-means, compared to density-based clustering algorithms such as DBSCAN. Note that in DBI higher values in the DBSCAN case mean worse clustering, as opposed to SC.

3.6.8 Calinski-Harabasz Index

The Calinski-Harabasz Index (CBI) [104] also known as the Variance Ratio Criterion, for a dataset E of size n_E with k clusters, is defined as:

$$\mathrm{CBI} = \frac{\mathrm{tr}\,(B_k)}{\mathrm{tr}\,(W_k)} \times \frac{n_E - k}{k - 1} \tag{3.25}$$

where $tr(B_k)$ is the trace of the between clusters dispersion matrix B_k and $tr(W_k)$ is the trace of the inside cluster dispersion W_k. The two matrices are defined as:

$$\begin{aligned} W_k &= \sum_{q=1}^{k} \sum_{x \in C_q} (x - c_q)\,(x - c_q)^T \\ B_k &= \sum_{q=1}^{k} n_q\,(c_q - c_E)\,(c_q - c_E)^T \end{aligned} \tag{3.26}$$

where C_q is the subset of data points in the cluster q, c_q is the centre of the cluster q, c_E the centre of the entire dataset E and n_q is the number of data points in the cluster q.

Higher CBI scores mean dense and well-defined clusters, which is the standard metric to evaluate a satisfying cluster. Like SC and DBI, CBI score will be generally higher for convex clustering algorithm such as K-means compared to density-based clustering algorithms such as DBSCAN.

3.6.9 Density-Based Clustering Validation Index

All the previous clustering performance metrics based on internal indexes fail at assessing the performance of density-based clustering algorithms such as DBSCAN, and do not take into account the noisy data points. The density-Based Clustering Validation Index (DBCV) [105] was created to overcome these issues and revolves around the idea of computing the density within a cluster, the density between clusters and shape properties, where high density within a cluster and low density

between clusters means an acceptable clustering. Some Density-based metrics were introduced before DBCV [106], some used the DBI [103] but they could not handle arbitrary shapes.

Given a dataset $O = \{o_1, ..., o_n\}$ containing n objects in an $\mathbb{R}^d$ space. Let KNN(o, i) be the distance between the data point o and its i^{th} nearest neighbour. Let $C = \{C_i, N\} \quad 1 \leq i \leq l$ be a clustering solution with l clusters and a possible empty set of noisy points N, where n_i is the size of the i^{th} cluster and n_N is the cardinality of noise.

The all-points-core-distance (a_{pts}coredist) which is the reverse of the density of each data point concerning all other data points inside its cluster, is defined as:

$$a_{pts}\text{coredist}(\mathbf{o}) = \left(\frac{\sum_{i=2}^{n_i} \left(\frac{1}{\text{KNN}(\mathbf{o}, i)} \right)^d}{n_i - 1} \right)^{-\frac{1}{d}} \tag{3.27}$$

The mutual reachability distance between two data points o_i and o_j is defined as $d_{mreach}(o_i, o_j) = max(a_{pts}\text{coredist}(o_i), a_{pts}\text{coredist}(o_j), d(o_i, o_j))$, where $d(o_i, o_j)$ is the distance between the two data points. The mutual reachability distance graph is a complete graph with the data points in O as vertices and the mutual reachability distance as weight for each edge. The minimum spanning tree of the graph is called MST$_{MRD}$. The overall idea behind DBCV is to compute the a_{pts}coredist and the MST$_{MRD}$ for each cluster C_i. Based on the minimum spanning tree computed before the density sparseness and density separation are used for the validity index.

The Density Sparseness of a Cluster (DSC) is defined as the maximum edge weight of the internal edges in the MST$_{MRD}$ for the cluster C_i. The Density Separation of a Pair of Cluster (DSPC) C_i and C_j with $1 \leq i, j \leq l, \quad i \neq j$ is defined as the minimum reachability distance between the internal nodes of the MST$_{MRD}$.

The validity index of a cluster is defined as:

$$V_C(C_i) = \frac{\min_{1 \leq j \leq l, j \neq i} (DSPC(C_i, C_j)) - DSC(C_i)}{\max\left(\min_{1 \leq j \leq l, j \neq i} (DSPC(C_i, C_j)), DSC(C_i)\right)} \tag{3.28}$$

The DBCV is defined as:

$$\text{DBCV}(C) = \sum_{i=1}^{i=l} \frac{|C_i|}{|O|} V_C(C_i) \tag{3.29}$$

The DBCV is a value between "-1" and "1" with higher values indicating better clustering solutions.

3.7 Logistic Network Lasso

The Logistic Network Lasso is a semi-supervised algorithm [6] taking advantage of partially labelled network-structured data, that can be applied in many domains such as image processing [107], social networks or bioinformatics [108]. The network-structured data is represented with a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \text{W})$ with every feedback $i \in \mathcal{V}$ being a node in the graph. Two feedback $i, j \in \mathcal{V}$ are connected by an undirected edge $\{i, j\} \in \mathcal{E}$ if they are considered similar, and each edge has an associated weight $W_{ij} > 0$ indicating the amount of similarity between the feedback. Each feedback $i \in \mathcal{V}$ has a binary label $y_i \in \{-1,1\}$ representing which cluster the meaning of the feedback fits in best. Being a semi-supervised algorithm these labels are going to be partly used in training the algorithm, with a sampling set $\mathcal{M} \subset \mathcal{V}$.

The labels y_i are modelled as independent random variables with unknown probabilities:

$$p_i = Prob\{y_i = 1\} \tag{3.30}$$

These probabilities are parametrized using the logarithm of the odds ratio, defined by the logarithm of the probability of success divided by the probability of failure:

$$x[i] = log\left(\frac{p_i}{1-p_i}\right) \tag{3.31}$$

The equation 3.31 defines a graph signal $x[\cdot] : \mathcal{V} \to \mathbb{R}$ used to represent the classifier that produces the labels $\hat{y}_i = sign(x[i])$. Minimizing the empirical error defined as:

$$\widehat{E}(x[\cdot]) := \frac{1}{|\mathcal{M}|}\sum_{i\in\mathcal{M}} \ell\left(y_i x[i]\right) \tag{3.32}$$

with the loss function ℓ:

$$\ell(z) = log(1 + exp(-z)) \tag{3.33}$$

is not enough to learn a classifier from the incomplete information provided by the sampled labels $y_i \in \mathcal{M}$. The equation 3.32 measures only how well the classifier can generate a predicted label that is equal to the correct label y_i. To use the information about the structure of the generated cluster, the total variation (TV) is defined:

$$\|x[\cdot]\|_{\text{TV}} := \sum_{\{i,j\}\in\mathcal{E}} W_{ij}|x[j] - x[i]| \tag{3.34}$$

The graph signal $x[\cdot]$ has a small TV if the values $x[i]$ are constant over well connected clusters. The TV and the empirical error are used to define the lnLasso problem:

$$\hat{x}[\cdot] \in \underset{x[\cdot]\in\mathbb{R}^{\mathcal{V}}}{\text{argmin}} \widehat{E}(x[\cdot]) + \lambda\|x[\cdot]\|_{\text{TV}} \tag{3.35}$$

The regularization parameter λ allows deciding how much the information gathered from the cluster structure should influence the classifier. The equation 3.35 does not enforce in a direct way to cluster the labels y_i, but they are forced to be clustered only as a result of minimizing the TV. The lnLasso problem is solved using the inexact form of the Alternating Direction Method of Multipliers (ADMM) [109].

ADMM solves problem that are in the form $minimize(f(x) + g(z))$ subject to $Ax + Bz = c$, with $f(x)$ and $g(z)$ convex functions. Defining the augmented Lagrangian as:

$$L_\rho(x, z, y) = f(x) + g(z) + y^T(Ax + Bz - c) + (\rho/2)\|Ax + Bz - c\|_2^2 \tag{3.36}$$

ADMM consists of the iterations:

$$\begin{aligned} x^{k+1} &:= \text{argmin}_x\, L_\rho\left(x, z^k, y^k\right) \\ z^{k+1} &:= \text{argmin}_z\, L_\rho\left(x^{k+1}, z, y^k\right) \\ y^{k+1} &:= y^k + \rho\left(Ax^{k+1} + Bz^{k+1} - c\right) \end{aligned} \tag{3.37}$$

with $\rho > 0$. In ADMM x and z are updated in an alternated manner, hence the name of the algorithm. The following conditions are necessary for the ADMM algorithm, the primal feasibility:

$$Ax^\star + Bz^\star - c = 0 \tag{3.38}$$

and the dual feasibility:

$$\begin{aligned} 0 &\in \partial f\left(x^\star\right) + A^T y^\star \\ 0 &\in \partial g\left(z^\star\right) + B^T y^\star \end{aligned} \tag{3.39}$$

Since z^{k+1} minimizes $L_\rho(x^{k+1}, z, y^k)$ the last equation of 3.39 is always satisfied for z^{k+1} and y^{k+1} reducing the total amount of equation to be satisfied for optimality to two. The primal residual is defined as:

$$r^{k+1} = Ax^{k+1} + Bz^{k+1} - c \tag{3.40}$$

and the dual residual as:

$$s^{k+1} = \rho A^T B \left(z^{k+1} - z^k\right) \tag{3.41}$$

The two residuals converge to zero when ADMM iterates, and they are used as the stopping criterion.

$$\left\|r^k\right\|_2 \leq \epsilon^{\text{pri}} \quad \text{and} \quad \left\|s^k\right\|_2 \leq \epsilon^{\text{dual}} \tag{3.42}$$

with $\epsilon^{pri} > 0$ and $\epsilon^{dual} > 0$. ADMM can be very slow to converge with high accuracy, but it is quick to converge with modest accuracy, usually in a few tens of iterations. In Machine Learning applications this improvement in accuracy after many iterations, generally do not provide prediction accuracy improvements and it is often skipped, stopping the algorithm after 20 or 30 iterations. Another way to relax the ADMM constraints is through inexact minimization if the functions are summable. Inexact ADMM works by solving the x and z minimization steps approximately at first, and more accurately as the iterations progress.

In order to apply ADMM the equation 3.35 needs to be reformulated in:

$$\hat{x}[\cdot] \in \underset{x[\cdot]\in\mathbb{R}^{\mathcal{V}}}{\operatorname{argmin}} \widehat{E}(x[\cdot]) + (\lambda/2) \sum_{(i,j)\in\mathcal{E}} W_{i,j} \left|z_{ij} - z_{ji}\right| \tag{3.43}$$

where z_{ij} is the auxiliary variable defined as $z_{ij} = x[i] \quad i \in \mathcal{V}, \quad j \in \mathcal{N}(i)$, with $\mathcal{N}(i)$ being defined as the neighbourhood of the node i: $\mathcal{N}(i) = \{j : \{i,j\} \in \mathcal{E}\}$.

The augmented Lagrangian is defined as:

$$\mathcal{L}_\rho\left(x[\cdot], z_{ij}, u_{ij}\right) = \widehat{E}(x[\cdot]) + (\lambda/2) \sum_{(i,j)\in\mathcal{E}} W_{ij} \left|z_{ij} - z_{ji}\right| + (\rho/2) \sum_{(i,j)\in\mathcal{E}} \left[\left(x[i] - z_{ij} + u_{ij}\right)^2 - u_{ij}^2\right] \tag{3.44}$$

where u_{ij} is the dual variable introduced for each edge $(i,j) \in \mathcal{E}$. The iterations to solve exact ADMM are formulated as:

$$\begin{aligned} x^{(k+1)}[\cdot] &:= \underset{x[\cdot]\in\mathbb{R}^{\nu}}{\operatorname{argmin}} \mathcal{L}_\rho\left(x[\cdot], z_{ij}^{(k)}, u_{ij}^{(k)}\right) \\ z_{ij}^{(k+1)} &:= \underset{z_{ij}\in\mathbb{R}}{\operatorname{argmin}} \mathcal{L}_\rho\left(x^{(k+1)}[\cdot], z_{ij}, u_{ij}^{(k)}\right) \\ u_{ij}^{(k+1)} &:= u_{ij}^{(k)} + x^{(k+1)}[i] - z_{ij}^{(k+1)} \text{ for each } (i,j) \in \mathcal{E} \end{aligned} \tag{3.45}$$

With the first equation minimizing the empirical error $\hat{E}(x[\cdot])$ and the second equation minimizing the total variation $\|x[\cdot]\|_{TV}$. The two equations are then coupled together in the last equation. The first equation of 3.45 can be rewritten knowing that the non sampled nodes $i \in \mathcal{M}$ have a label $\tilde{y}_i = 0$, otherwise $\tilde{y}_i \in \{-1{,}1\}$:

$$x^{(k+1)}[i] = \underset{x\in\mathbb{R}}{\operatorname{argmin}} \ell\left(\tilde{y}_i x\right) + \frac{|\mathcal{M}|\rho}{2} \sum_{j\in\mathcal{N}(i)} \left(x - z_{ij}^{(k)} + u_{ij}^{(k)}\right)^2 \tag{3.46}$$

The second equation of 3.45 can be rewritten as:

$$z_{ij}^{(k+1)} = \theta \left(x^{(k+1)}[i] + u_{ij}^{(k)} \right) + (1-\theta) \left(x^{(k+1)}[j] + u_{ji}^{(k)} \right) \tag{3.47}$$

with

$$\theta = \max \left(\frac{1}{2}, 1 - \frac{(\lambda/\rho) W_{ij}}{\left| x^{(k+1)}[i] + u_{ij}^{(k)} - x^{(k+1)}[j] - u_{ji}^{(k)} \right|} \right) \tag{3.48}$$

Using inexact approximation 3.46 can be approximated as:

$$\hat{x}^{(k+1)}[i] = \underbrace{\Phi_i \circ \ldots \circ \Phi_i}_{|\tilde{y}_i| \lceil 2 \log(2(k+1)) / \log(|\mathcal{M}| \rho d_i) \rceil} (1/d_i) \sum_{j \in \mathcal{N}(i)} \left(z_{ij}^{(k)} - u_{ij}^{(k)} \right) \tag{3.49}$$

with d_i being the node degree $d_i = |\mathcal{N}(i)|$ and the map defined as:

$$\Phi_i(x) := \frac{\tilde{y}_i / \left(|\mathcal{M}| d_i \rho \right)}{1 + \exp\left(\tilde{y}_i x \right)} + (1/d_i) \sum_{j \in \mathcal{N}(i)} \left(z_{ij}^{(k)} - u_{ij}^{(k)} \right) \tag{3.50}$$

The classifier $x[i]$ once trained is able to label the data points as $\hat{y}_i = 1$ if $x[i] > 0$ and $\hat{y}_i = -1$ if $x[i] < 0$. The magnitude of the classifier $|x[i]|$ gives a measure of the confidence of the prediction, with higher values meaning higher confidence.

Chapter 4

Results

The first research question is what is the most effective way to encode the sentence meaning for the downstream task of binary clustering feedback with a certain degree of similarity. From the set of techniques analyzed in the chapter Encoding feedback, the Word2Vec, Doc2Vec, FastText, GloVe and BERT models were chosen for comparison. The Word2Vec, FastText and Doc2Vec models were implemented using Gensim [110] training them on the dataset of the thesis. For the Word2Vec model, both the CBOW and Skip-Gram variation were used, with negative sampling and a window of five tokens. The FastText model was implemented with $n = 5$ and $n = 10$ iterations. The Doc2Vec model was implemented with both PV-DM and PV-DBOW variations. For all the three models, the dimension of the sentence vector was kept to 100 for a better comparison. The GloVe model was implemented downloading the pre-trained embeddings, more precisely those with a dimension of 100, pre-trained on the Gigaword corpus + Wikipedia 2014 [15].

Due to better documentation and also since the implementation of ALBERT and RoBERTa variations is continuously changing to improve their rank in the GLUE leaderboard [67], the vanilla BERT implementation was chosen. As presented in the paragraph about BERT, there are two different model sizes, for the scope of the thesis the smaller model Bert-Base has been used.

To obtain the sentence embedding with BERT, first the model is fine-tuned using MRPC [68]. Since sentence embedding are not directly supported by BERT, as suggested by one of the authors of the paper, the model is trained with the feedback and the last hidden layer and the second-last hidden layer values are extracted for each token, obtaining a 728-dimensional vector. Each token-vector is added into a 728-dimensional sentence vector that will be divided by the number of token-vectors used. This sentence embedding is the feature vector x that is used to define the Machine Learning problem as explained in the paragraph The Machine Learning problem. To evaluate the performance, tests have been done using only the last layer, using only the second-last layer and using an average of both the layers. Since BERT uses two unique tokens CLS and SEP for fine-tuning, those vectors were removed from the sentence embedding to avoid extra noise.

BERT classifier can be used to give a score of semantic similarity between sentences. Each feedback is paired with all the remaining feedback from the round, generating all the possible combinations. For each combination, scores ranging from zero to one are computed, where higher values mean higher similarity. These scores are pruned with different thresholds, and only the scores above the threshold are selected to be the weights between the nodes in the lnLasso implementation. (See the paragraph Logistic Network Lasso).

For NMI and AMI, the average method is kept to the default which is arithmetic. For AP the preference was selected as the best in 100 possible values that generated two clusters for each round, in terms of the metrics evaluated. For SPC with Nearest Neighbour (SPC_NN), the Nearest Neighbour is selected from 100 possible values that provide the best overall result, for each round. For SPC with Radial Basis Function (SPC_RBF) the gamma is selected from 300 possible values that provided the best overall result, for each round. For DBSCAN, when clustering evaluation metrics that relied on external indexes (RI, ARI, NMI, AMI, FMI, V-measure) were used, special *eps* and *min_sample* values were selected in order to allow only two clusters to exist with no

limit to the number of noisy data points. The clustering evaluation metrics that use external indexes do not take into account the presence of noisy points. To adjust the results for this type of evaluation metrics the number of noisy data points for DBSCAN was limited to be less than 80, and those labels were then randomly assigned to the two clusters.

For all the clustering solutions except DBSCAN, the labels that were equal to zero were removed from both the true labels and the predicted labels. For DBSCAN those labels were kept to see if the algorithm could predict these values, but on average the correct noise points predicted was below 5%, leading in lower results of RI, ARI and FMI due to a bigger chance to fail while predicting.

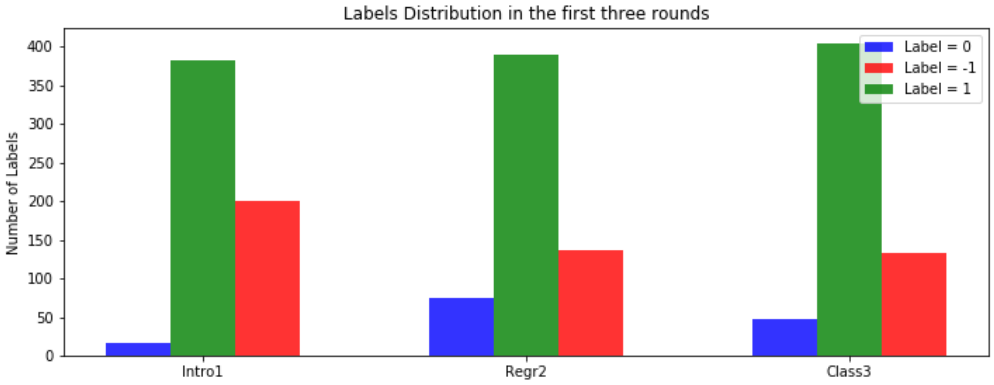

Figure 4.1. Figure showing the labels distribution in the three rounds, where a strong unbalance between $label = -1$ and $label = 1$ can be noticed.

As we can see in Figure 4.1, the labels are very unbalanced toward $label = 1$ meaning that the clustering performance metric RI will not give a clear, unbiased result. This unbalance on the feedback can be explained in terms of when the feedback questionnaire was asked to be filled. Most of the students filled the questionnaire immediately after the quiz that was related mostly on the exercise session, so students were unconsciously nudged into giving a feedback that was related to some problem that they experienced shortly before the quiz, compared to some other problem that they experienced while studying during the lecture the week before. A solution to solve this problem of unbalanced labels would be to either downsample the more frequent class, or introduce synthetic data points in the more infrequent class. Since the amount of data for each round is not in the order of ten thousand, adding synthetic data points is the suggested approach [111]. Since an unbalance in classes is pretty common in a real-world scenario and the thesis aim was to test the algorithms with a dataset that was as less synthetic as possible, new data points were not introduced.

All the following tables are obtained using the tokenized feedback with little pre-processing (no stemming). This has been done to see if there was an improvement in accuracy using stemming, as shown in the paragraph Sentence embeddings with stemming.

4.1 Comparison of different clustering methods with different sentence embeddings using external indexes metrics

Round	Clustering Algorithm	**Word2Vec** (CBOW)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.6021	**0.0249**	0.0063	0.0048	0.5752	0.0063	0.0064
Regr2		0.6376	0.0088	0.0005	-0.0012	0.6321	0.0005	0.0005
Class3		0.7263	**0.0871**	0.0229	0.0180	0.7138	0.0215	0.0233
Intro1	AP	**0.6566**	0.0044	0.0171	0.0010	**0.7367**	0.0039	0.0063
Regr2		**0.7381**	-0.0025	0.0064	-0.0009	**0.7818**	0.0015	0.0024
Class3		/	/	/	/	/	/	/
Intro1	SPC_NN	0.5160	0.0009	0.0043	0.0029	0.5241	0.0044	0.0043
Regr2		0.6546	0.0243	0.0030	0.0012	0.6446	0.0029	0.0031
Class3		0.6816	0.0575	0.0119	0.0099	0.6601	0.0118	0.0112
Intro1	SPC_RBF	0.6329	-0.0191	0.0364	0.0135	0.7175	0.0206	0.0297
Regr2		0.7078	**0.0559**	0.0115	0.0079	0.7042	0.0107	0.0117
Class3		**0.7493**	0.0459	0.0156	0.0092	**0.7690**	0.0131	0.0155
Intro1	DBSCAN	0.5533	-0.0452	**0.0429**	**0.0324**	0.6322	**0.0386**	**0.0438**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6832	-0.0721	**0.0361**	**0.0257**	0.7120	**0.0319**	**0.0368**

Table 4.1. Results obtained using the Word2Vec sentence encoding with the CBOW variation. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. For the third round, the AP algorithm never converged with only two clusters. The best value for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**Word2Vec** (Skip-gram)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5561	0.0100	0.0012	-0.0002	0.5886	0.0012	0.0013
Regr2		0.7400	0	**1**	0	**0.7839**	0	0
Class3		0.6778	-0.0058	0.0001	0.0001	0.6867	0.0001	0.0001
Intro1	AP	/	/	/	/	/	/	/
Regr2		0.7400	0	**1**	0	0.7839	0	0
Class3		/	/	/	/	/	/	/
Intro1	SPC_NN	**0.6449**	-0.0052	0.0017	-0.0007	0.7232	0.0010	0.0014
Regr2		**0.7469**	**0.0425**	0.0343	0.0133	0.7773	0.0206	0.0289
Class3		**0.7609**	**0.0484**	0.0534	0.0185	**0.7882**	0.0118	0.0409
Intro1	SPC_RBF	0.5674	0.0137	0.0018	0.0003	0.6190	0.0017	0.0019
Regr2		0.6964	0.0337	0.0056	0.0031	0.6959	0.0052	0.0058
Class3		0.7549	0.0325	0.0200	0.0065	0.7841	0.0115	0.0165
Intro1	DBSCAN	0.0640	**0.7589**	**0.662**	**0.6515**	**0.8907**	**0.6618**	**0.6686**
Regr2		0.6478	-0.0967	0.0773	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6901	-0.0794	**0.0571**	**0.0400**	0.7202	**0.0493**	**0.0579**

Table 4.2. Results obtained using the Word2Vec sentence encoding with the Skip-gram variation. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. For the first and third round, the AP algorithm never converged with only two clusters. The best value for each round and for each column are marked in **bold**.

For the Word2Vec embeddings generated with the Skip-gram variation (Table 4.2), the clustering algorithms produced unreliable results, where the perfect scores of NMI in K-means and AP (when it converged) are justified as being perfectly randomly generated labels. The high scores in DBSCAN are generated by noisy solutions (ca. 90% of the total). For this reason, the Word2Vec Skip-gram embeddings were not used for the next comparison.

Round	Clustering Algorithm	**Doc2Vec** (PV-DM)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5966	**0.0223**	0.0056	0.0042	0.5695	0.0056	0.0057
Regr2		0.6319	0.0179	0.0020	0.0003	0.6196	0.0020	0.0019
Class3		0.7393	0.0838	0.0239	0.0017	0.7372	0.0214	0.0244
Intro1	AP	0.6123	-0.0121	0.0018	-0.0001	0.6648	0.0015	0.0018
Regr2		0.685	0.0314	0.0040	0.0018	0.6821	0.0038	0.0041
Class3		0.7486	**0.0872**	0.0291	0.0194	0.7516	0.0250	0.0295
Intro1	SPC_NN	0.6148	0.0194	0.0057	0.0040	0.6136	0.0055	0.0057
Regr2		0.5104	0.0070	0.0034	0.0017	0.5595	0.0034	0.0033
Class3		0.5374	0.0127	0.0094	0.0071	0.5658	0.0033	0.0091
Intro1	SPC_RBF	**0.6346**	-0.0178	0.0347	0.0123	**0.7192**	0.0190	0.0019
Regr2		**0.7442**	**0.0513**	0.0254	0.0117	**0.7686**	0.0177	0.0058
Class3		**0.7568**	0.0814	**0.0344**	0.0200	**0.7683**	0.0271	0.0165
Intro1	DBSCAN	0.5657	-0.0294	**0.0465**	**0.0223**	0.7085	**0.0312**	**0.0421**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6918	-0.0693	0.0329	**0.0229**	0.7190	**0.0288**	**0.0335**

Table 4.3. Results obtained using the Doc2Vec sentence encoding with the PV-DM variation. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**Doc2Vec** (PV-DBOW)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5022	-0.0024	0.0006	-0.0007	0.5241	0.0006	0.0006
Regr2		0.7495	0.0349	0.0737	0.0206	**0.7853**	0.0323	0.0494
Class3		0.7616	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	AP	/	/	/	/	/	/	/
Regr2		/	/	/	/	/	/	/
Class3		/	/	/	/	/	/	/
Intro1	SPC_NN	**0.6861**	**0.0716**	**0.0554**	**0.0332**	**0.7150**	**0.0434**	0.0545
Regr2		**0.7761**	**0.2240**	**0.1095**	**0.0954**	0.7423	**0.1045**	**0.1115**
Class3		**0.7709**	0.0963	0.0727	**0.0371**	0.7874	**0.0509**	**0.0679**
Intro1	SPC_RBF	0.4991	-0.0018	0.0006	-0.0007	0.5227	0.0006	0.0006
Regr2		0.7571	0.1557	0.0674	0.0533	0.7404	0.0618	0.0687
Class3		0.7482	**0.1030**	0.0355	0.0258	0.7443	0.0317	0.0362
Intro1	DBSCAN	/	/	/	/	/	/	/
Regr2		0.6927	0.0571	0.0132	0.0097	0.6876	0.0123	0.0135
Class3		0.7209	0.0894	0.0305	0.0225	0.7169	0.0274	0.0311

Table 4.4. Results obtained using the Doc2Vec sentence encoding with the PV-DBOW variation. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. AP and the first round in DBSCAN never converged with two clusters for all the parameters tested. The best values for each round and for each column are marked in **bold**.

For the Doc2Vec (PV-DBOW) implementation (Table 4.4) the unreliability was even worse compared to Word2Vec Skip-gram, affecting also DBSCAN. For the same reason as the Word2Vec Skip-gram, Doc2Vec PV-DBOW embeddings were not used for the next comparison.

Round	Clustering Algorithm	**FastText** ($n = 5$)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5530	0.0090	0.0011	-0.0003	0.5812	0.0011	0.0011
Regr2		0.6509	0.0485	0.0115	0.0097	0.6232	0.0116	0.0114
Class3		0.7225	**0.0930**	0.0252	0.0208	0.7048	0.0241	0.0255
Intro1	AP	0.6123	-0.0005	5.81e-7	-0.0013	0.6449	5.25e-7	5.93e-7
Regr2		0.7268	**0.0855**	0.0249	0.0185	0.7206	0.0228	0.0254
Class3		0.7579	0.0869	0.0372	0.0222	0.7680	0.02967	0.0369
Intro1	SPC_NN	0.5005	-0.0013	0.0005	-0.0008	0.5234	0.0005	0.0004
Regr2		**0.7343**	0.0690	0.0217	0.0135	**0.7427**	0.0183	0.0219
Class3		0.6369	0.0581	0.0243	0.0210	0.6020	0.0249	0.0238
Intro1	SPC_RBF	**0.6419**	**0.0094**	0.0047	0.0004	0.6930	0.0024	0.0033
Regr2		0.7283	0.0684	0.0203	0.0144	0.7317	0.0184	0.0207
Class3		**0.7635**	0.0783	0.0467	0.0241	**0.7812**	0.0336	0.0442
Intro1	DBSCAN	0.625	-0.0329	**0.0511**	**0.0249**	**0.7044**	**0.0348**	**0.0470**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6901	-0.0794	**0.0571**	**0.0400**	0.7202	**0.0493**	**0.0579**

Table 4.5. Results obtained using the FastText sentence encoding with $n = 5$ iterations. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**FastText** ($n = 10$)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5866	**-0.0007**	2.08e-6	-0.0013	0.5976	2.01e-6	2.12e-6
Regr2		0.5905	0.0337	0.0054	0.0037	0.6329	0.0054	0.0054
Class3		0.7300	0.0613	0.0139	0.0091	0.7318	0.0124	0.0142
Intro1	AP	0.6192	-0.0042	0.0002	-0.0012	0.6653	0.0001	0.0002
Regr2		0.7173	**0.0647**	0.0154	0.0107	0.7153	0.0140	0.0157
Class3		0.7561	**0.0950**	0.0373	0.0240	0.7613	0.0311	0.0375
Intro1	SPC_NN	0.5074	-0.0012	0.0005	-0.0008	0.5244	0.0005	0.0005
Regr2		0.5070	-0.0013	0.0016	0.0001	0.5542	0.0017	0.0016
Class3		0.7143	0.0403	0.0346	0.0114	0.7193	0.0174	0.0246
Intro1	SPC_RBF	**0.6312**	-0.0011	6.49e-06	-0.0013	0.6858	5.20e-06	6.45e-6
Regr2		**0.7268**	0.0341	0.0079	0.0037	**0.7465**	0.0065	0.0079
Class3		**0.7654**	0.0871	0.0522	0.0276	**0.7810**	0.0380	0.04972
Intro1	DBSCAN	0.6267	-0.0318	**0.0498**	**0.0238**	**0.7060**	**0.0334**	**0.0454**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0789**
Class3		0.6901	-0.0794	**0.0571**	**0.0400**	0.7202	**0.0493**	**0.0579**

Table 4.6. Results obtained using the FastText sentence encoding with $n = 10$ iterations. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Between the two FastText implementation with $n = 5$ (Table 4.5) and $n = 10$ (Table 4.6) iterations, FastText with $n = 5$ iterations was selected for further experiments, due to better results most of the times, as we can see in Table 4.7. Table 4.7 was computed by doing the average of the three rounds for the best results of each table 4.5 and 4.6.

Embedding method	**Comparison of different FastText embeddings**				
	RI	ARI	NMI	AMI	FMI
FastText n=5	**0.7132**	**0.0626**	**0.0618**	**0.0427**	0.7428
FastText n=10	0.7078	0.0530	0.0614	0.0423	**0.7445**

Table 4.7. Comparison of the two embeddings generated by FastText with $n = 5$ and $n = 10$ iterations. The best value for each column is marked in **bold**.

Round	Clustering Algorithm	**GloVe** (100d Gigaword corpus + Wikipedia 2014)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5883	0.0091	0.0011	-0.0003	0.5812	0.0011	0.0011
Regr2		0.6907	0.0720	0.0177	0.0151	0.6671	0.0174	0.0179
Class3		0.7598	0.0860	0.0396	0.0229	0.7716	0.0309	0.0390
Intro1	AP	0.5633	**0.0093**	0.0009	-0.0005	0.6132	0.0009	0.001
Regr2		0.7438	**0.0984**	0.0374	0.0257	0.7450	0.0323	0.0380
Class3		0.7654	0.0950	0.0521	0.0294	0.7779	0.0396	0.0507
Intro1	SPC_NN	0.4957	0.0002	0.0020	0.0007	0.5230	0.0020	0.0020
Regr2		0.5655	0.0157	0.0161	0.0133	0.5609	0.0165	0.0157
Class3		0.5367	0.0140	0.013	0.0103	0.5658	0.0134	0.0126
Intro1	SPC_RBF	**0.6419**	0.0087	0.0046	0.0003	**0.6941**	0.0023	0.0031
Regr2		**0.7571**	0.0629	**0.1053**	0.0391	**0.7866**	0.0570	**0.0834**
Class3		**0.7709**	**0.1043**	**0.0692**	0.0379	**0.7842**	**0.0510**	**0.0664**
Intro1	DBSCAN	0.5933	-0.0456	**0.0511**	**0.0320**	0.6734	**0.0411**	**0.0508**
Regr2		0.6478	-0.0967	0.0773	**0.0632**	0.6832	**0.0719**	0.0788
Class3		0.6901	-0.0794	0.0571	**0.0400**	0.7202	0.0493	0.0579

Table 4.8. Results obtained using the GloVe sentence encoding with 100-dimensional vectors pre–trained on the Gigaword corpus and Wikipedia 2014. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**BERT** (Last-Layer Embeddings)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5623	0.0313	0.0103	0.0089	0.5653	0.0103	0.0103
Regr2		0.5825	0.0123	0.0025	0.0009	0.5755	0.0026	0.0025
Class3		0.7426	**0.0716**	0.0208	0.0133	0.7482	0.0178	0.0210
Intro1	AP	0.6175	0.0456	0.0187	0.0173	0.5665	0.0187	0.0187
Regr2		0.5097	0.0038	0.0285	0.0247	0.5605	0.0292	0.0279
Class3		0.5598	0.0213	0.0052	0.0034	0.5873	0.0053	0.0051
Intro1	SPC_NN	**0.6895**	**0.0690**	**0.0864**	**0.0446**	0.7301	**0.0604**	**0.0806**
Regr2		**0.7590**	**0.0814**	**0.0754**	**0.0351**	0.7818	**0.0493**	**0.0657**
Class3		**0.7654**	0.0573	0.0723	**0.0264**	0.7925	**0.0393**	**0.0574**
Intro1	SPC_RBF	0.6504	-0.0045	0.0145	0.0015	**0.7349**	0.0046	0.0074
Regr2		0.7495	0.0349	0.0737	0.0206	**0.7853**	0.0323	0.0494
Class3		0.7616	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	DBSCAN	0.5400	-0.0430	0.0485	0.0371	0.6267	0.0438	0.0495
Regr2		0.6512	0.0635	0.0220	0.0192	0.6175	0.0214	0.0222
Class3		0.6575	0.0429	0.0089	0.0067	0.6432	0.0086	0.0090

Table 4.9. Results obtained using the BERT sentence embedding with 728-dimensional vectors pre-trained on MRPC corpus [68], using only the last-layer of the model. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**BERT** (Second Last-Layer Embeddings)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.6141	0.0372	0.0121	0.0105	0.5762	0.0120	0.0121
Regr2		0.7359	0.0149	0.0057	0.0513	0.7013	0.0554	0.0576
Class3		0.7456	0.1343	0.0469	0.0394	0.7246	0.0445	0.0478
Intro1	AP	0.6244	0.0511	0.0210	0.0196	0.5721	0.0210	0.0210
Regr2		0.6224	-0.0061	0.0001	-0.0015	0.6241	0.0001	0.0001
Class3		0.7542	**0.1551**	0.0592	**0.0503**	0.7303	**0.0563**	**0.0603**
Intro1	SPC_NN	**0.6827**	**0.0517**	**0.1053**	**0.0452**	**0.7363**	**0.0636**	**0.0900**
Regr2		**0.7761**	**0.1872**	**0.1017**	**0.0761**	0.7634	**0.0902**	**0.1035**
Class3		**0.7616**	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	SPC_RBF	0.6504	-0.0045	0.0145	0.0015	0.7349	0.0046	0.0074
Regr2		0.7495	0.0349	0.0737	0.0206	**0.7853**	0.0323	0.0494
Class3		**0.7616**	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	DBSCAN	0.5650	-0.0410	0.0366	0.0258	0.6452	0.0319	0.0372
Regr2		0.6827	0.0808	0.0273	0.0227	0.6609	0.0259	0.0277
Class3		0.6927	0.0803	0.0252	0.0201	0.6809	0.0236	0.0257

Table 4.10. Results obtained using the BERT sentence embedding with 728-dimensional vectors pre-trained on MRPC corpus [68], using only the second last-layer of the model. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best values for each round and for each column are marked in **bold**.

Round	Clustering Method	**BERT**(Average Last and second-last layers)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.6034	0.0310	0.0102	0.0088	0.5656	0.0101	0.0102
Regr2		0.7218	0.1078	0.0339	0.0288	0.6992	0.0325	0.0345
Class3		0.5404	0.0039	0.0022	0.0006	0.5624	0.0023	0.0021
Intro1	AP	0.6329	**0.0539**	0.0198	0.0179	0.5892	0.0196	0.0199
Regr2		0.4956	-0.0114	0.0268	0.0234	0.5628	0.0274	0.0263
Class3		0.6257	0.0301	0.0062	0.0044	0.6066	0.0063	0.0062
Intro1	SPC_NN	**0.6792**	0.0450	**0.0968**	**0.0392**	**0.7365**	**0.0559**	**0.0803**
Regr2		**0.7742**	**0.1491**	0.1073	**0.0662**	0.7793	**0.0848**	**0.1060**
Class3		0.7616	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	SPC_RBF	0.6055	0.0377	0.0164	0.0149	0.5553	0.0165	0.0164
Regr2		0.7628	0.0838	**0.1260**	0.0532	**0.7879**	0.0752	**0.1068**
Class3		**0.7672**	**0.0735**	0.0661	**0.0290**	0.7896	**0.0416**	0.0580
Intro1	DBSCAN	/	/	/	/	/	/	/
Regr2		/	/	/	/	/	/	/
Class3		/	/	/	/	/	/	/

Table 4.11. Results obtained using the BERT sentence embedding with 728-dimensional vectors pre-trained on MRPC corpus [68], using the average of the last and second last-layers of the model. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. DBSCAN converged only with a massive amount of noise points (ca. 90%). The best values for each round and for each column are marked in **bold**.

Embedding method	**Comparison of different BERT layers embedding**				
	RI	ARI	NMI	AMI	FMI
BERT Last Layer	0.7380	0.0765	0.0794	0.0354	0.7712
BERT Second Last Layer	0.7101	**0.1313**	0.0945	**0.0572**	0.7717
BERT Average Layers	**0.7402**	0.0922	**0.0997**	0.0448	**0.7726**

Table 4.12. Comparison of external clustering metric performances using different combination of BERT layers. The best value for each column is marked in **bold**.

Results from Table 4.12 are the average of the three rounds of the best results for each Table 4.9, 4.10 and 4.11. The average of the two BERT layers will produce results that are slightly better than both the embeddings generated when using only the last layer and only the second-last layer, at the expense of a higher computational complexity when computing the feature matrix (usually a 5x time increase). In the case of this thesis, with a total amount of feedback of ca. three thousand, computing all feature matrix on a laptop CPU (i7 9750H) would take on average ca. 20 minutes, which is still a feasible time for the aim to tackle eventual problems quickly. For the next comparison, only the BERT embeddings using the second-last layer and the average of both layers will be used, because they provided on average better results.

4.1.1 Sentence embeddings with stemming

As discussed in the paragraph stemming, stemming is a popular pre-processing technique that increases the accuracy and improves the computational time for NLP models. This chapter is dedicated into investigating if the stemming is a necessary pre-processing technique for increasing the binary clustering feedback accuracy.

Round	Clustering Algorithm	**Word2Vec** (CBOW STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5780	**0.0126**	0.0027	0.0014	0.5564	0.0027	0.0027
Regr2		0.6243	0.0209	0.0029	0.0013	0.6090	0.0030	0.0029
Class3		0.7151	0.0756	0.0177	0.0140	0.7012	0.0169	0.0180
Intro1	AP	0.6123	0.0114	0.0013	-0.0002	0.6234	0.0012	0.0134
Regr2		0.7078	0.0639	0.0143	0.0105	0.6997	0.0013	0.0146
Class3		/	/	/	/	/	/	/
Intro1	SPC_NN	0.5321	0.0026	0.0032	0.0019	0.5240	0.0033	0.0032
Regr2		0.512	0.0094	0.0038	0.0021	0.5622	0.0039	0.0037
Class3		0.6294	0.0144	0.0034	0.0017	0.6314	0.0033	0.0034
Intro1	SPC_RBF	**0.6398**	-0.0137	0.0291	0.0088	**0.7246**	0.0145	0.0216
Regr2		**0.7495**	**0.0552**	0.0392	0.0169	**0.7763**	0.0251	0.0348
Class3		**0.7564**	**0.0956**	0.0379	0.0244	**0.7617**	0.0316	0.0381
Intro1	DBSCAN	0.6317	-0.0284	**0.0455**	**0.0205**	0.7108	**0.0293**	**0.0405**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6901	-0.0794	**0.0571**	**0.0400**	0.7202	**0.0493**	**0.0578**

Table 4.13. Results obtained using the Word2Vec sentence encoding with the CBOW variation and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. For the third round, the AP algorithm never converged with only two clusters. The best value for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**Doc2Vec**(PV-DM STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5901	**0.0233**	0.0077	0.0063	0.5545	0.0027	0.0027
Regr2		0.6490	0.0356	0.0062	0.0045	0.6291	0.0030	0.0029
Class3		0.7076	**0.0693**	0.0152	0.0121	0.6927	0.0169	0.0180
Intro1	AP	0.5664	0.0085	0.0008	-0.0007	0.6251	0.0007	0.0008
Regr2		0.7135	0.0600	0.0133	0.0092	0.7114	0.0122	0.0136
Class3		0.7467	0.0081	0.0259	0.0169	0.7511	0.0222	0.0262
Intro1	SPC_NN	0.5431	0.0184	0.0138	0.0120	0.5320	0.0139	0.0136
Regr2		0.6125	0.0156	0.0033	0.0015	0.6011	0.0033	0.0032
Class3		0.7419	0.0575	0.0190	0.0117	0.7527	0.0161	0.0191
Intro1	SPC_RBF	0.6226	0.0025	0.0001	-0.0013	0.6611	0.0001	0.0001
Regr2		**0.7495**	**0.0703**	0.0385	0.0199	**0.7694**	0.0280	0.0367
Class3		**0.7654**	0.0662	**0.0592**	0.0249	**0.7892**	0.0363	0.0511
Intro1	DBSCAN	**0.6250**	-0.0329	**0.0511**	**0.0249**	**0.7044**	**0.0348**	**0.0470**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6866	-0.0813	0.0588	**0.0418**	0.7170	**0.0512**	**0.0597**

Table 4.14. Results obtained using the Doc2Vec sentence encoding with the PV-DM variation and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best value for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**FastText** ($n = 5$ STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5540	0.0106	0.0015	0.0001	0.5811	0.0014	0.0015
Regr2		0.6319	0.0245	0.0036	0.0019	0.6153	0.0036	0.0036
Class3		0.7058	0.0564	0.0106	0.0077	0.6961	0.0101	0.0108
Intro1	AP	0.6089	0.0020	0.0001	-0.0013	0.6335	0.0001	0.0001
Regr2		0.7192	**0.0754**	0.0196	0.0145	0.7127	0.0181	0.0200
Class3		0.7337	0.0628	0.0151	0.0097	0.7327	0.0133	0.0154
Intro1	SPC_NN	0.4864	-0.0014	0.0002	-0.0011	0.5260	0.0002	0.0002
Regr2		0.6099	-0.0012	0.0026	0.0009	0.6198	0.0026	0.0026
Class3		0.5203	0.0010	0.0016	0.0001	0.5662	0.0016	0.0015
Intro1	SPC_RBF	**0.6460**	**0.0125**	0.0026	0.0002	0.6981	0.0020	0.0025
Regr2		**0.7431**	0.0373	0.0211	0.0080	**0.7728**	0.0130	0.0177
Class3		**0.7598**	**0.0782**	0.0381	0.0208	**0.7748**	0.0287	0.0369
Intro1	DBSCAN	0.6317	-0.0284	**0.0455**	**0.0205**	**0.7108**	**0.0293**	**0.0405**
Regr2		0.6478	-0.0967	**0.0773**	**0.0632**	0.6832	**0.0719**	**0.0788**
Class3		0.6901	-0.0794	**0.0571**	**0.0400**	0.7202	**0.0493**	**0.0579**

Table 4.15. Results obtained using the FastText sentence encoding with $n = 5$ iterations and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best value for each round and for each column are marked in **bold**.

Round	Clustering Algorithm	**GloVe**(Gigaword+Wikipedia STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5461	0.0066	0.0008	-0.0005	0.5667	0.0008	0.0008
Regr2		0.6983	0.0807	0.0212	0.0182	0.6742	0.0207	0.0215
Class3		0.7374	0.0812	0.0225	0.0159	0.7352	0.0202	0.0229
Intro1	AP	0.5729	**0.0092**	0.0017	0.0004	0.5554	0.0017	0.0017
Regr2		0.7457	**0.0854**	0.0354	0.0219	0.7551	0.0289	0.0354
Class3		0.7579	**0.0906**	0.0381	0.0234	0.7664	0.0309	0.0380
Intro1	SPC_NN	0.5060	0.0030	0.0014	0.0001	0.5363	0.0014	0.0014
Regr2		0.7457	0.0210	0.0544	0.0114	0.7847	0.0196	0.0308
Class3		0.5587	0.0124	0.0137	0.0110	0.5648	0.0142	0.0134
Intro1	SPC_RBF	**0.6432**	-0.0108	0.0250	0.0064	**0.7277**	0.0113	0.0172
Regr2		**0.7590**	0.0738	**0.0912**	0.0380	**0.7853**	0.0544	**0.0773**
Class3		**0.7672**	0.0778	**0.0627**	0.0290	**0.7880**	0.0411	0.0565
Intro1	DBSCAN	0.5717	-0.0243	**0.0398**	**0.0173**	0.7155	**0.0252**	**0.0348**
Regr2		0.6495	-0.0924	0.0623	**0.0509**	0.6833	**0.0581**	0.0636
Class3		0.6918	-0.0784	0.0563	**0.0391**	0.7218	**0.0484**	**0.0570**

Table 4.16. Results obtained using the GloVe sentence encoding with 100-dimensional vectors pre-trained on the Gigaword corpus and Wikipedia 2014, and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best value for each round and for each column are marked in **bold**.

Round	Embedding method	**BERT** (Second-Layer STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5108	0.0071	0.0023	0.0010	0.5345	0.0024	0.0023
Regr2		0.5256	0.0005	0.0006	-0.0008	0.5548	0.0007	0.0006
Class3		0.5523	0.0148	0.0034	0.0017	0.5812	0.0035	0.0033
Intro1	AP	0.5815	**0.0120**	0.0023	0.0009	0.5632	0.0022	0.0023
Regr2		0.6376	0.0404	0.0094	0.0076	0.6115	0.0096	0.0094
Class3		0.4935	0.0019	0.0013	-0.0002	0.5611	0.0013	0.0012
Intro1	SPC_NN	0.5039	0.0004	0.0013	0	0.5233	0.0013	0.0013
Regr2		**0.7583**	**0.0670**	**0.1091**	**0.0419**	**0.7869**	**0.0606**	**0.0878**
Class3		**0.7616**	**0.0377**	**0.0764**	**0.0215**	**0.7934**	**0.0336**	**0.0514**
Intro1	SPC_RBF	**0.6587**	0.0062	**0.0302**	**0.0039**	**0.7392**	**0.0084**	**0.0135**
Regr2		0.7514	0.0419	0.0822	0.0252	0.7856	0.0385	0.0582
Class3		**0.7616**	**0.0377**	**0.0764**	**0.0215**	**0.7934**	**0.0336**	**0.0514**
Intro1	DBSCAN	/	/	/	/	/	/	/
Regr2		/	/	/	/	/	/	/
Class3		/	/	/	/	/	/	/

Table 4.17. Results obtained using the BERT sentence embedding with 728-dimensional vectors pre-trained on MRPC corpus [68], using only the second last-layer of the model, and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best value for each round and for each column are marked in **bold**.

Round	Embedding method	**BERT**(Average Last and Second-last layers STEMMING)						
		RI	ARI	NMI	AMI	FMI	V05	V15
Intro1	K-Means	0.5105	0.0057	0.0016	0.0003	0.5355	0.0017	0.0016
Regr2		0.5165	0.0011	0.0063	0.0044	0.5549	0.0065	0.0062
Class3		0.5829	0.0112	0.0021	0.0005	0.5805	0.0022	0.0021
Intro1	AP	0.6312	**0.0642**	**0.0358**	**0.0337**	0.5622	**0.0360**	**0.0356**
Regr2		0.6433	0.0490	0.0133	0.0112	0.6128	0.0134	0.0131
Class3		0.4976	-0.0008	0.0005	-0.0009	0.5589	0.0005	0.0005
Intro1	SPC_NN	0.5458	0.0171	0.0061	0.0047	0.5467	0.0061	0.0060
Regr2		**0.7556**	**0.0573**	**0.0990**	**0.0354**	**0.7864**	**0.0521**	**0.0766**
Class3		0.7616	0.0377	**0.0764**	0.0215	**0.7934**	0.0336	0.0514
Intro1	SPC_RBF	**0.6587**	0.0062	0.0302	0.0039	**0.7392**	0.0084	0.0135
Regr2		0.7495	0.0349	0.0737	0.0206	0.7853	0.0323	0.0494
Class3		**0.7654**	**0.0618**	0.0639	**0.0252**	0.7908	**0.0372**	**0.0533**
Intro1	DBSCAN	/	/	/	/	/	/	/
Regr2		/	/	/	/	/	/	/
Class3		/	/	/	/	/	/	/

Table 4.18. Results obtained using the BERT sentence embedding with 728-dimensional vectors pre-trained on MRPC corpus [68], using an average of the last and the second last-layer of the model, and pre-processing with stemming. All results are averaged after five iterations. V05 and V15 are the V-measure scores using beta 0.5 and 1.5, respectively. SPC_NN stands for SPC with Nearest Neighbour. SPC_RBF stands for SPC with Radial Basis Function. The best value for each round and for each column are marked in **bold**.

In the table 4.17 for the third row Class3 SPC_NN and SPC_RBF have the same score, meaning that they converged at the same solution. In both table 4.17 and 4.18 DBSCAN never converged to an useful solution, and the values for the 90% noisy points solutions were omitted.

Embeddings method	**Comparison of different embedding methods without and with stemming**				
	RI	ARI	NMI	AMI	FMI
Word2Vec	0.7147	0.0560	0.0521	0.0404	0.7625
Word2Vec Stemming	0.7152	0.0545	0.0600	0.0469	0.7542
Doc2Vec	0.7119	0.0536	0.0527	0.0361	0.7520
Doc2Vec Stemming	0.7133	0.0543	0.0625	0.0433	0.7543
FastText	0.7132	0.0626	0.0618	0.0427	0.7428
FastText Stemming	0.7163	0.0554	0.0600	0.0409	0.7528
GloVe	0.7233	0.0707	0.0752	0.0451	0.7539
GloVe Stemming	0.7231	0.0617	0.0646	0.0358	0.7670
Bert SL	0.7101	**0.1313**	0.0945	**0.0572**	0.7717
BERT SL Stemming	0.7262	0.0389	0.0719	0.0224	**0.7732**
Bert BL	**0.7402**	0.0922	**0.0997**	0.0448	0.7726
BERT BL Stemming	0.7266	0.0611	0.0704	0.0314	0.7730

Table 4.19. Comparison of the different embeddings method with and without stemming for pre processing the input. BERT SL stands for BERT using only the second-last layer, and BERT BL stands for BERT using the average of the last and second-last layers. The best value for each clustering evaluation metric is marked in **bold**.

The final comparison shown in Table 4.19, is made using Table 4.1, 4.13, 4.4, 4.13, 4.5, 4.15, 4.8, and Table 4.16, averaging the best values for each evaluation metric, for Intro1, Regr2 and Class3. The final comparison table 4.19 shows that GloVe and BERT were more affected negatively by stemming, compared to the other embeddings method. BERT, in particular, is even more penalized by the stemming compared to GloVe, in both cases using only the second-last layer and the average of both last and second-last layers. For the other embedding methods stemming did not provide a clear improvement in the metrics evaluated (in some cases it produced a decrease) that justified the extra step in the NLP pipeline. For this reason, the embeddings without stemming were used for the next comparisons.

4.2 Comparison of different clustering methods with different sentence embeddings using internal indexes metrics

For the SC, DBI and CBI scores in the DBSCAN case, the noisy points were limited to be less than 80 data points (maximum of noisy points in the first three rounds), and those labels were then randomly assigned to the two clusters. This has been done to avoid different results when comparing SC, DBI and CBI, that already penalize density-based clustering on their own. Even if there was no constraint on the noisy data points other than to be less than 80, DBSCAN preferred to produce solutions with very few noisy data points (on average less than five noisy points), because they were better in term of the metrics evaluated.

In order to compute the DBCV, the implementation from Christopher Jennes has been used (url), but contrary to what presented in the original paper [105] this implementation is extremely slow and can't handle noisy data points, hence the random assignments that were performed for SC, DBI and CBI was also applied here. Most of the times, the DBCV implementation did not converge to a solution, making it impossible to compare the numerous different embedding methods. For this reason, the DBCV metric was left out of the clustering performance metrics analysis.

Note that as stated in the chapter about the DBI, lower values of DBI indicate better clustering.

Round	Clustering Algorithm	**Word2Vec CBOW**			**Doc2Vec PV-DM**		
		SC	DBI	CBI	SC	DBI	CBI
Valid4	K-Means	0.5873	0.6501	**674.77**	0.6354	0.6003	**812.35**
Clust5		0.5652	0.6752	**553.26**	0.5844	0.6286	**668.90**
Featlear6		0.6825	0.5630	**890.23**	0.6746	0.5539	**890.44**
Valid4	AP	0.636	0.6051	624.12	0.6504	0.5667	753.23
Clust5		0.6628	0.6249	454.59	0.6180	0.5848	564.25
Featlear6		/	/	/	0.6848	0.5299	888.22
Valid4	SPC_NN	0.6547	0.5678	572.65	0.5804	0.5575	566.64
Clust5		0.4846	0.6911	452.61	0.3576	0.5742	73.29
Featlear6		0.4423	0.7658	441.93	0.6607	0.5801	874.64
Valid4	SPC_RBF	0.5795	0.6556	673.45	0.6014	0.6325	786.32
Clust5		0.5820	0.6656	551.89	0.5649	0.6483	659.55
Featlear6		0.6693	0.5849	870.51	0.6566	0.5860	866.70
Valid4	DBSCAN	**0.7889**	**0.1568**	126.32	**0.7513**	**0.1928**	120.49
Clust5		**0.8103**	**0.4088**	211.24	**0.7836**	**0.5067**	157.98
Featlear6		**0.7406**	**0.2468**	312.55	**0.7166**	**0.4596**	272.02

Table 4.20. Results obtained using the Word2Vec CBOW variation and Doc2Vec PV-DM variation sentence embeddings. All results are averaged after five iterations. The best values for each round, for each column and for each type of sentence embeddings are marked in **bold**.

Round	Clustering Algorithm	**GloVe**			**FastText**(n=5)		
		SC	DBI	CBI	SC	DBI	CBI
Valid4	K-Means	0.6354	0.6431	**633.08**	0.5868	0.6719	**553.92**
Clust5		0.7092	0.5952	**659.55**	0.5912	0.6688	**514.80**
Featlear6		0.7268	0.5220	**782.65**	0.6727	0.6303	**620.48**
Valid4	AP	0.7247	0.5318	564.99	0.7251	0.5404	458.94
Clust5		0.7339	0.5334	631.61	0.6989	**0.5433**	414.43
Featlear6		0.7430	0.4855	746.62	0.7543	0.4823	511.04
Valid4	SPC_NN	0.4773	0.7107	426.97	0.4964	0.6941	455.66
Clust5		0.4773	1.0329	469.32	0.5853	0.6723	514.60
Featlear6		0.3824	1.1054	148.27	0.4616	0.7619	380.47
Valid4	SPC_RBF	**0.8304**	**0.1621**	324.78	0.6454	0.6244	497.18
Clust5		0.7940	**0.3193**	554.33	0.623	0.6372	486.24
Featlear6		0.771	**0.3476**	470.36	0.7046	0.6008	530.74
Valid4	DBSCAN	0.8219	0.3497	222.38	**0.8275**	**0.4705**	171.52
Clust5		**0.8024**	0.5201	127.24	**0.8174**	1.3978	75.64
Featlear6		**0.7963**	0.4929	165.29	**0.8146**	**0.1671**	307.52

Table 4.21. Results obtained using the GloVe pre-trained word vectors, trained on the Gigaword corpus + Wikipedia 2014 and the FastText model with $n = 5$ iterations embeddings. All results are averaged after five iterations. The best values for each round, for each column and for each type of sentence embeddings are marked in **bold**.

Round	Clustering Algorithm	**BERT**(Last-layer)			**BERT**(Second last-layer)		
		SC	DBI	CBI	SC	DBI	CBI
Valid4	K-Means	0.2176	2.9136	**48.80**	0.1422	3.5282	**40.72**
Clust5		0.1782	3.3445	**36.02**	0.1532	3.3693	**35.62**
Featlear6		0.2198	2.9171	**44.52**	0.1589	3.3477	**39.69**
Valid4	AP	/	/	/	0.0619	4.4652	21.56
Clust5		0.1180	3.3003	30.31	0.0773	3.6287	29.80
Featlear6		0.1577	3.3114	37.84	0.1003	4.2936	25.30
Valid4	SPC_NN	0.2147	2.7881	31.49	0.3267	**0.5404**	9.56
Clust5		0.1848	**1.2160**	7.73	0.2314	1.6090	9.35
Featlear6		**0.2832**	**0.9853**	13.88	0.2807	**0.9348**	14.55
Valid4	SPC_RBF	**0.4222**	**1.2508**	14.94	**0.4699**	0.9588	12.39
Clust5		0.1158	2.4852	30.71	**0.4071**	**0.7770**	6.34
Featlear6		0.0883	2.2063	25.79	**0.4323**	1.1667	13.73
Valid4	DBSCAN	0.2699	3.0759	15.07	0.4320	1.4408	15.11
Clust5		**0.2620**	3.1981	14.88	0.2965	2.7330	7.41
Featlear6		0.2471	3.2702	15.37	0.3713	1.5850	9.34

Table 4.22. Results obtained using the last-layer and the second-last layer of BERT trained on MRPC corpus [68]. All results are averaged after five iterations. The best values for each round, for each column and for each type of sentence embeddings are marked in **bold**.

Round	Clustering Algorithm	**BERT**(Average last and second-last layers)		
		SC	DBI	CBI
Valid4	K-Means	0.1934	3.1701	**42.40**
Clust5		0.0505	3.5464	**35.75**
Featlear6		0.1845	3.2153	**40.44**
Valid4	AP	0.0646	4.4805	21.58
Clust5		0.1009	3.5659	29.81
Featlear6		0.1373	3.6819	30.44
Valid4	SPC_NN	0.2845	1.0963	9.61
Clust5		0.2368	1.9636	9.77
Featlear6		0.2784	**0.9567**	14.18
Valid4	SPC_RBF	0.3415	**0.5253**	10.07
Clust5		0.3026	1.0928	7.06
Featlear6		**0.4344**	1.2033	13.40
Valid4	DBSCAN	**0.4678**	1.1936	15.32
Clust5		**0.4624**	**0.4072**	5.63
Featlear6		0.4002	1.0979	9.58

Table 4.23. Results obtained using the average of the last-layer and the second-last layer of BERT trained on MRPC corpus [68]. All results are averaged after five iterations. The best values for each round, for each column and for each type of sentence embeddings are marked in **bold**.

Embeddings method	**Comparison of different embeddings methods**		
	SC	DBI	CBI
Word2Vec	0.7799	**0.2708**	706.09
Doc2Vec	0.7505	0.3864	**790.56**
GloVe	0.8097	0.2763	691.76
FastText	**0.8198**	0.3936	563.06
BERT LL	0.3225	1.1654	43.11
BERT SL	0.4364	0.7507	38.68
BERT BL	0.4549	0.6297	39.53

Table 4.24. Comparison of the different embeddings method in terms of internal indexes metrics. BERT LL stands for BERT using only the last layer, BERT SL stands for BERT using only the second-last layer, and BERT BL stands for BERT using the average of the last and second-last layers. The best value for each clustering evaluation metric is marked in **bold**.

As explained in SC, DBI and CBI all tend to penalize DBSCAN, but we can see from the tables 4.20 and 4.21 that SC and DBI tend to penalize less DBSCAN compared to CBI. Even with this penalization, DBSCAN still produced the best clustering solution in terms of the metrics evaluated in table 4.20 and 4.21. Table 4.24 was computed using the average the best results of each table. As we can see from Table 4.24 in terms of internal indexes metrics BERT does not give good results, probably due to the higher dimensionality of the embeddings (728d vs 100d), even if it produces the best solutions in terms of external indexes metrics.

For this reason, internal indexes metrics proved to be not reliable when assessing the performance of the different clustering algorithms with different embeddings, and they will not be further analyzed in the next experiments.

4.3 Logistic Network Lasso for binary clustering

For the next results the V-Measure scores with $\beta = 0.5$ and $\beta = 1.5$ were excluded on purpose, because from the previous experiment there was no justification into focusing more on the homogeneity of a clustering solution compared to completeness and vice versa. As explained in Logistic Network Lasso, lnLasso is a semisupervised algorithm that uses a percentage of the true label to improve the learning of the classifier. The main parameters of lnLasso are the regularization parameter λ and the percentage of the true labels that are used. For this round of experiments, all the weights of the graph of the network are set to 1, meaning that every feedback is connected with each other. The main focus of this round of experiment is to show how the clustering metrics change at the modification of the regularization parameter and the percentage of the true labels used.

Round	**Word2Vec** (DBOW)						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.5857	0.0333	0.0138	0.0105	0.5444	0.1	0.001
Intro1	**0.6280**	**0.0672**	**0.0353**	**0.0304**	0.5646	0.2	0.001
Intro1	0.5817	0.0173	0.0057	0.0022	0.5684	0.1	0.01
Intro1	0.6257	0.0564	0.0208	0.0151	**0.6030**	0.2	0.01
Intro1	0.5963	0.0218	0.0055	0.0020	0.5879	0.1	0.1
Intro1	0.5940	0.0187	0.0041	0.0010	0.5951	0.2	0.1
Regr2	0.5804	0.0489	0.0115	0.0072	0.5670	0.1	0.001
Regr2	0.6193	**0.0881**	**0.0334**	**0.0247**	0.5924	0.2	0.001
Regr2	0.5904	0.0265	0.0070	0.0032	0.5879	0.1	0.01
Regr2	**0.6196**	0.0436	0.0146	0.0080	0.6197	0.2	0.01
Regr2	0.5897	0.0133	0.0052	0.0013	0.6004	0.1	0.1
Regr2	0.6176	0.0362	0.0075	0.0033	**0.6376**	0.2	0.1
Class3	0.6408	0.0521	0.0157	0.0102	0.6256	0.1	0.001
Class3	0.6421	**0.0823**	**0.0259**	**0.0194**	0.6139	0.2	0.001
Class3	0.6226	0.0401	0.0101	0.0061	0.6085	0.1	0.01
Class3	**0.6747**	0.0701	0.0212	0.0121	**0.6720**	0.2	0.01
Class3	0.6312	0.0043	0.0031	-0.0004	0.6438	0.1	0.1
Class3	0.6455	0.0162	0.0063	0.0024	0.6529	0.2	0.1

Table 4.25. Results obtained running the lnLasso algorithm with Word2Vec DBOW variation embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Round	**Doc2Vec** (PV-DM)						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.5793	0.0297	0.0111	0.0079	0.5547	0.1	0.001
Intro1	**0.6347**	**0.0824**	**0.0394**	**0.0341**	0.5725	0.2	0.001
Intro1	0.5890	0.0313	0.0113	0.0069	0.5696	0.1	0.01
Intro1	0.5947	0.0324	0.0099	0.0065	0.5696	0.2	0.01
Intro1	0.5827	0.0188	0.0047	0.0017	0.5756	0.1	0.1
Intro1	0.5947	0.0270	0.0095	0.0054	**0.5870**	0.2	0.1
Regr2	0.5455	0.0290	0.0094	0.0060	0.5409	0.1	0.001
Regr2	0.5900	**0.0609**	**0.0220**	**0.0164**	0.5616	0.2	0.001
Regr2	0.5701	0.0312	0.0059	0.0028	0.5656	0.1	0.01
Regr2	0.6196	0.0527	0.0127	0.0070	0.6244	0.2	0.01
Regr2	0.5674	0.0114	0.0041	0.0011	0.5840	0.1	0.1
Regr2	**0.6272**	0.0149	0.0055	0.0008	**0.6581**	0.2	0.1
Class3	0.6151	0.0383	0.0110	0.0067	0.6005	0.1	0.001
Class3	0.6668	**0.1027**	**0.0397**	**0.0293**	0.6384	0.2	0.001
Class3	0.6134	0.0246	0.0055	0.0014	0.6072	0.1	0.01
Class3	**0.6774**	0.0345	0.0125	0.0049	**0.6908**	0.2	0.01
Class3	0.6257	0.0033	0.0026	-0.0002	0.6331	0.1	0.1
Class3	0.6637	0.0142	0.0058	0.0003	0.6815	0.2	0.1

Table 4.26. Results obtained running the lnLasso algorithm with Doc2Vec PV-DM variation embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Round	**FastText** ($n = 5$)						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.5703	0.0305	0.0156	0.0122	0.5348	0.1	0.001
Intro1	**0.6420**	**0.0879**	**0.0444**	**0.0390**	0.5720	0.2	0.001
Intro1	0.5903	0.0352	0.0157	0.0122	0.5488	0.1	0.01
Intro1	0.6233	0.0540	0.0224	0.0175	0.5844	0.2	0.01
Intro1	0.5957	0.0239	0.0070	0.0037	0.5876	0.1	0.1
Intro1	0.6157	0.0219	0.0066	0.0022	**0.6307**	0.2	0.1
Regr2	0.5681	0.0296	0.0084	0.0049	0.5589	0.1	0.001
Regr2	0.6193	**0.0768**	**0.0255**	**0.0176**	0.6011	0.2	0.001
Regr2	0.5797	0.0375	0.0104	0.0066	0.5697	0.1	0.01
Regr2	0.6000	0.0420	0.0114	0.0067	0.5972	0.2	0.01
Regr2	0.5831	0.0061	-0.1981	0	0.5959	0.1	0.1
Regr2	**0.6425**	0.0022	-0.1915	0.0006	**0.6871**	0.2	0.1
Class3	0.6154	0.0435	0.0147	0.0091	0.5977	0.1	0.001
Class3	**0.6733**	**0.1332**	**0.0510**	**0.0506**	0.6360	0.2	0.001
Class3	0.6387	0.0469	0.0128	0.0081	0.6249	0.1	0.01
Class3	0.6616	0.0517	0.0164	0.0099	0.6550	0.2	0.01
Class3	0.6315	0.0254	0.0056	0.0022	0.6347	0.1	0.1
Class3	0.6729	0.0389	0.0166	0.0074	**0.6866**	0.2	0.1

Table 4.27. Results obtained running the lnLasso algorithm with FastText with $n = 5$ iterations embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Round	**GloVe**						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.6003	0.0391	0.0159	0.0123	0.5595	0.1	0.001
Intro1	**0.6343**	**0.0783**	**0.0388**	**0.0337**	0.5685	0.2	0.001
Intro1	0.5690	0.0267	0.0101	0.0065	0.5452	0.1	0.01
Intro1	0.6037	0.0467	0.0242	0.0204	0.5489	0.2	0.01
Intro1	0.5827	0.0162	0.0040	0.0012	0.5732	0.1	0.1
Intro1	0.6117	0.0325	0.0096	0.0054	**0.6050**	0.2	0.1
Regr2	0.5860	0.0424	0.0131	0.0083	0.5793	0.1	0.001
Regr2	**0.6030**	**0.0756**	**0.0250**	**0.0185**	0.5772	0.2	0.001
Regr2	0.5704	0.0265	0.0104	0.0064	0.5579	0.1	0.01
Regr2	0.5977	0.0336	0.0097	0.0056	0.5981	0.2	0.01
Regr2	0.5831	-0.002	0.0047	0.0014	0.6027	0.1	0.1
Regr2	0.5957	0.0283	0.0071	0.0025	**0.6198**	0.2	0.1
Class3	0.6068	0.0467	0.0153	0.0106	0.5850	0.1	0.001
Class3	0.6548	0.0918	0.0312	**0.0235**	0.6248	0.2	0.001
Class3	0.6154	0.0479	0.0120	0.0076	0.5980	0.1	0.01
Class3	0.6695	**0.0996**	**0.0321**	0.0223	0.6535	0.2	0.01
Class3	0.6257	0.0112	0.0035	0.0001	0.6268	0.1	0.1
Class3	**0.6740**	0.0156	0.0192	0.0033	**0.7035**	0.2	0.1

Table 4.28. Results obtained running the lnLasso algorithm with GloVe pre-trained word vectors, trained on the Gigaword corpus + Wikipedia 2014 embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Round	**BERT** (Second-last layer)						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.5730	0.0253	0.0119	0.0088	0.5372	0.1	0.001
Intro1	0.6200	0.0663	0.0336	0.0292	0.5586	0.2	0.001
Intro1	0.6067	0.0476	0.0178	0.0136	0.5675	0.1	0.01
Intro1	**0.6477**	**0.0856**	**0.0457**	**0.0389**	**0.5894**	0.2	0.01
Intro1	0.6020	0.0385	0.0144	0.0107	0.5690	0.1	0.1
Intro1	0.6043	0.0407	0.0155	0.0119	0.5706	0.2	0.1
Regr2	0.5874	0.0465	0.0102	0.0057	0.5795	0.1	0.001
Regr2	0.6276	0.0869	0.0350	0.0252	0.6015	0.2	0.001
Regr2	0.5787	0.0435	0.0111	0.0072	0.5643	0.1	0.01
Regr2	0.6372	**0.0962**	**0.0393**	**0.0278**	0.6154	0.2	0.01
Regr2	0.5821	0.0366	0.0115	0.0065	0.5824	0.1	0.1
Regr2	**0.6475**	0.0349	0.0296	0.0093	**0.6734**	0.2	0.1
Class3	0.6281	0.0566	0.0139	0.0093	**0.6989**	0.1	0.001
Class3	0.6599	**0.1037**	**0.0414**	**0.0327**	0.6230	0.2	0.001
Class3	0.6274	0.0644	0.0206	0.0152	0.6020	0.1	0.01
Class3	0.6449	0.0873	0.0331	0.0262	0.6106	0.2	0.01
Class3	0.6397	0.0347	0.0100	0.0049	0.6370	0.1	0.1
Class3	**0.6716**	0.0346	0.0112	0.0045	0.6832	0.2	0.1

Table 4.29. Results obtained running the lnLasso algorithm with using the second-last layer of BERT embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Round	**BERT**(Average last and second-last layers)						
	RI	ARI	NMI	AMI	FMI	PROB	REG
Intro1	0.6000	0.0373	0.0136	0.0101	0.5661	0.1	0.001
Intro1	0.6203	0.0674	0.0305	0.0261	0.5623	0.2	0.001
Intro1	0.5943	0.0393	0.0134	0.0096	0.5672	0.1	0.01
Intro1	**0.6370**	**0.0819**	**0.0373**	**0.0319**	0.5769	0.2	0.01
Intro1	0.6087	0.0395	0.0139	0.010	0.5813	0.1	0.1
Intro1	0.6263	0.0460	0.0193	0.0138	**0.6099**	0.2	0.1
Regr2	0.5804	0.0388	0.0109	0.0064	0.5695	0.1	0.001
Regr2	0.6203	**0.0988**	**0.0396**	**0.0301**	0.5905	0.2	0.001
Regr2	0.5754	0.0199	0.0079	0.0033	0.5719	0.1	0.01
Regr2	0.6189	0.0898	0.0332	0.0247	0.5914	0.2	0.01
Regr2	0.6159	0.0208	0.0079	0.0022	0.6329	0.1	0.1
Regr2	**0.6405**	0.0513	0.0159	0.0073	**0.6610**	0.2	0.1
Class3	0.6459	0.0513	0.0139	0.0089	0.6348	0.1	0.001
Class3	0.6531	0.1075	0.0378	0.0303	0.6184	0.2	0.001
Class3	0.6134	0.0428	0.0113	0.0075	0.5933	0.1	0.01
Class3	0.6805	**0.1266**	**0.0520**	**0.0389**	0.6497	0.2	0.01
Class3	0.6592	0.0444	0.0110	0.0051	0.6620	0.1	0.1
Class3	**0.6866**	0.0349	0.0193	0.0061	**0.7097**	0.2	0.1

Table 4.30. Results obtained running the lnLasso algorithm with using the average of the last and second-last layers of BERT embeddings. PROB stands for probability and indicates the percentage of true labels used, (i.e. 0.1 = 10%) and reg is the regularization parameter λ. The best result for each metric and for each round is marked in **bold**.

Embeddings method	**Comparison of different embedding methods**				
	RI	ARI	NMI	AMI	FMI
Word2Vec	0.6298	0.0792	0.0315	0.0248	0.5903
Doc2vec	0.6305	0.0820	0.0337	0.0266	0.5908
FastText	**0.6526**	0.0993	0.0403	**0.0357**	0.6030
Glove	0.6356	0.0845	0.0320	0.0348	0.5997
BERT SL	0.6483	0.0952	0.0421	0.0331	**0.6093**
BERT BL	0.6459	**0.1024**	**0.0430**	0.0336	0.6057

Table 4.31. Comparison of running the lnLasso algorithm using different embeddings. The best result for each clustering performance metric is marked in **bold**.

Table 4.31 was computed using the average of the best values for all the three rounds, for each different embeddings method. As we can see from Table 4.31, the best results were obtained using the FastText and BERT embeddings, which will be further analyzed in the next paragraph.

4.3.1 Comparison using BERT similarity score

In order to truly let the lnLasso algorithm work at its peak performance, it has to be applied on a network structured dataset where part of the data points in the graph are connected by weighted edges. To obtain the weighted edges that connect similar feedbacks, BERT scores of semantic similarity were computed between each feedback forming every possible combination. The scores were pruned with different thresholds of 0.6, 0.7, 0.8 and 0.9, and for each pruning the lnLasso algorithm was executed on the feedbacks encoded with different embeddings. Only the three embeddings that obtained the best scores in the previous comparison were selected, FastText with $n = 5$ iterations, BERT using only the second layer and BERT using the average of the last and second-last layer.

Round	**BERT**(Second-last layer)					
	RI	ARI	NMI	AMI	FMI	SIM
Intro1	0.6967	0.1171	0.0980	0.0655	0.6856	0.9
Intro1	**0.6983**	**0.1443**	**0.0882**	**0.0719**	**0.6425**	**0.8**
Intro1	0.6983	0.1280	0.0909	0.0634	0.6804	0.7
Intro1	0.6817	0.1103	0.0613	0.0450	0.6565	0.6
Regr2	0.6844	0.1080	0.0938	0.0539	0.6788	0.9
Regr2	**0.6827**	**0.1323**	**0.0796**	**0.0463**	**0.6857**	**0.8**
Regr2	0.6628	0.0788	0.0427	0.0230	0.6701	0.7
Regr2	0.6844	0.1006	0.0943	0.0474	0.6929	0.6
Class3	0.7295	0.1467	0.0992	0.0563	0.7283	0.9
Class3	0.7209	0.1535	0.0780	0.0501	0.7097	0.8
Class3	0.7192	0.0917	0.0869	0.0400	0.7299	0.7
Class3	**0.7260**	**0.1151**	**0.1075**	**0.0581**	**0.7236**	**0.6**

Table 4.32. Results obtained running the lnLasso algorithm using the second-last layer of BERT embeddings. SIM stands for the threshold of similarity scores. The row with the best results for each round is marked in **bold**.

Round	**BERT**(Average last and second-last layers)					
	RI	ARI	NMI	AMI	FMI	SIM
Intro1	0.6983	0.1160	0.1099	0.0695	0.6959	0.9
Intro1	**0.7117**	**0.1443**	**0.1294**	**0.0876**	**0.6951**	**0.8**
Intro1	0.6667	0.0916	0.0457	0.0344	0.6361	0.7
Intro1	0.6817	0.1161	0.0678	0.0545	0.6321	0.6
Regr2	0.6811	0.1113	0.0808	0.0478	0.6744	0.9
Regr2	0.6777	0.0864	0.0786	0.0403	0.6842	0.8
Regr2	0.6794	0.0814	0.0917	0.0411	0.6958	0.7
Regr2	**0.6927**	**0.1302**	**0.1067**	**0.0591**	**0.6926**	**0.6**
Class3	0.7243	0.1023	0.1083	0.0532	0.7291	0.9
Class3	0.7243	0.1608	0.0896	0.0607	0.7029	0.8
Class3	0.7226	0.1222	0.0845	0.0476	0.7197	0.7
Class3	**0.7312**	**0.1406**	**0.1066**	**0.0602**	**0.7272**	**0.6**

Table 4.33. Results obtained running the lnLasso algorithm using the average of the last and second-last layers of BERT embeddings. SIM stands for the threshold of similarity scores. The row with the best results for each round is marked in **bold**.

Round	**FastText**($n = 5$)					
	RI	ARI	NMI	AMI	FMI	SIM
Intro1	**0.7133**	**0.1420**	**0.1571**	**0.0986**	**0.7085**	**0.9**
Intro1	0.7033	0.1288	0.1116	0.0749	0.6901	0.8
Intro1	0.6917	0.1120	0.0989	0.0591	0.7023	0.7
Intro1	0.6850	0.1048	0.0731	0.0465	0.6877	0.6
Regr2	0.6811	0.0865	0.0933	0.0435	0.6946	0.9
Regr2	0.6744	0.1024	0.0665	0.0393	0.6698	0.8
Regr2	**0.6827**	**0.1017**	**0.0865**	**0.0461**	**0.6862**	**0.7**
Regr2	0.661	0.0806	0.0529	0.0298	0.6669	0.6
Class3	**0.7329**	**0.1228**	**0.1476**	**0.0695**	**0.7404**	**0.9**
Class3	0.7295	0.1483	0.1004	0.0620	0.7167	0.8
Class3	0.7329	0.1366	0.1240	0.0710	0.7247	0.7
Class3	0.7072	0.1167	0.0683	0.0457	0.6864	0.6

Table 4.34. Results obtained running the lnLasso algorithm using the FastText with $n = 5$ iterations embeddings. SIM stands for the threshold of similarity scores. The row with the best results for each round is marked in **bold**.

By looking at tables 4.32, 4.33 and 4.34 it seems that there is no clear correlation between the threshold used to prune the edges, and the best results that are marked in bold. Another round of experiments has been done selecting the threshold that provided the best results for each different embeddings, and instead of directly using the BERT score for similarity as the weight of the edge, a value of 100 has been given as a weight to the edges that had a BERT score above the threshold and a weight of 1 to the other edges, similar to what has been done in [112]. Unfortunately, this weighting schema did not provide reliable results, with ARI and AMI scores that were always 0 or very close to 0, therefore the results were not reported.

Embeddings method	**Comparison of different embedding methods using lnLasso**				
	RI	ARI	NMI	AMI	FMI
BERT SL	0.7023	0.1306	0.0918	0.0588	0.6839
BERT BL	**0.7119**	**0.1384**	0.1142	0.0690	0.7050
FastText	0.7096	0.1222	**0.1304**	**0.0714**	**0.7117**

Table 4.35. Comparison of running the lnLasso algorithm with BERT similarity scores as weighting scheme using different embeddings. The best results for each clustering performance metric are marked in **bold**.

Table 4.35 was computed using the average of the best rows for all the three rounds for each embedding. Comparing table 4.31 and table 4.35 we can see that the BERT similarity scores weighting scheme is needed to have a substantial increase in accuracy, in terms of the clustering performance metrics evaluated.

Embedding Method	**Comparison of different clustering algorithm using different embeddings with lnLasso**				
	RI	ARI	NMI	AMI	FMI
Word2Vec	0.7147	0.0560	0.0521	0.0404	0.7625
Doc2Vec	0.7119	0.0536	0.0527	0.0361	0.7520
FastText	0.7132	0.0626	0.0618	0.0427	0.7428
GloVe	0.7233	0.0707	0.0752	0.0451	0.7550
BERT SL	0.7101	0.1313	0.0945	0.0572	0.7717
BERT BL	**0.7402**	0.0922	0.0997	0.0448	**0.7726**
lnLasso BERT SL	0.7023	0.1306	0.0918	0.0588	0.6839
lnLasso BERT BL	0.7119	**0.1384**	0.1142	0.0690	0.7050
lnLasso FastText	0.7096	0.1222	**0.1304**	**0.0714**	0.7117

Table 4.36. Comparison of the raw different clustering algorithms and the lnLasso algorithm using different embeddings. The best results for each clustering performance metric are marked in **bold**.

Table 4.36 was computed using the average of the best values for the three rounds, for the raw clustering algorithms analyzed in the previous paragraphs (K-means, AP, SPC_NN, SPC_RBF and DBSCAN) and the lnLasso algorithm. As we can see from table 4.36, the lnLasso algorithm improved ARI, NMI and AMI, especially when using the FastText embeddings.

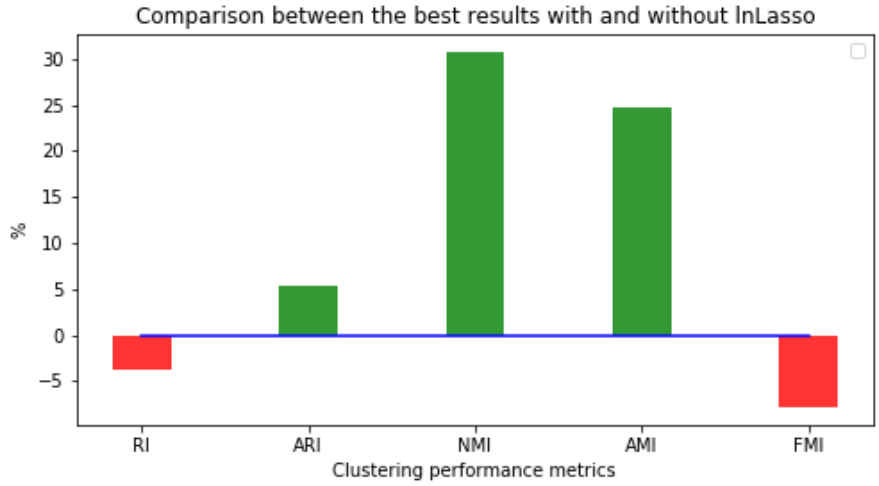

Figure 4.2. Bar plot showing the increment or decrement in percentage between the best result for each clustering performance metrics, comparing the result obtained with and without lnLasso.

The Bar plot 4.2 shows how in terms of ARI, NMI, and AMI lnLasso provides an improvement of up to 30%. RI and FMI scores are better without lnLasso.

Chapter 5

Conclusions and future work

The efficiency of LnLasso has been researched in binary classification scenarios with partially labelled networked data. For the first time, in this thesis, the performance of the lnLasso algorithm was tested when binary-clustering feedback data.

An overview is given of the different embedding methods that can be used using standard clustering methods and in combination with lnLasso. The sets of experiments executed in this thesis showed that the best embedding methods from the ones analyzed for binary clustering feedback data are GloVe, FastText and BERT. After extensive analysis of which clustering method is the most optimal while clustering feedback data, Spectral Clustering proved to be the one providing the best results majority of the time. Stemming, even if it is a typical pre-processing step, decreased the performance of the clustering algorithm using the best embedding methods. The clustering performance metrics that used internal indexes proved to be unreliable and could not assess the performance of BERT, most probably due to the higher dimensionality of the vectors. For this reason, if the true labels are available, an analysis with the clustering performance metrics that used external indexes is preferred.

One round of experiments proved that assigning weights to the edges that connect similar feedback is a necessary step for lnLasso. Results using BERT similarity scores as weights for the edges that connect comparable feedback indicate that BERT is a suitable method for identifying similar feedback for lnLasso.

As explained in the paragraph Logistic Network Lasso, lnLasso is a semi-supervised algorithm that uses a percentage of the true labels during training. For this reason, a certain improvement over the other unsupervised algorithms that were tested was expected, proportional to the percentage of true labels used. Using a rate of 20% of true labels, lnLasso increased the best value of ARI, NMI and AMI of up to 30%, but decreased RI and FMI. The best combination of embedding method to use in conjunction with lnLasso is either FastText or BERT using the average of both second-last and last-layer. The results of the experiments made in the previous chapter indicate that lnLasso is a valid choice for tackling the problem of binary clustering feedback data.

The research that has been done in this thesis can be further extended on multiple points. In order to better compare the lnLasso algorithm, the experiments were done in terms of binary clustering. Binary clustering might not always be an efficient approach when clustering feedback data and depends on how the questions were formulated in the first place. LnLasso can be reformulated for non binary classification problems or multiclass classification problems, and then compared to the clustering techniques analyzed in this thesis without forcing to only have two clusters. For the particular dataset used in this thesis, feedback was rarely ambiguous and could be easily labelled into one of the three clusters. For this reason, only hard-clustering techniques were analyzed, but this assumption might not work for all datasets. For multiclass classification problems, the lnLasso needs to be compared to soft-clustering techniques. In order to further automate the process, topic modelling can be used to identify the classes for a multiclass approach.

Bibliography

[1] Phil Long and George Siemens, https://er.educause.edu/articles/2011/9/penetrating-the-fog-analytics-in-learning-and-education

[2] I. Khan and A. Pardo, "Data2u: Scalable real time student feedback in active learning environments", Proceedings of the Sixth International Conference on Learning Analytics & Knowledge, New York, NY, USA, 2016, pp. 249–253, DOI 10.1145/2883851.2883911

[3] M. Jovanovic, M. Vukicevic, M. Milovanovic, and M. Minovic, "Using data mining on student behavior and cognitive style data for improving e-learning systems: a case study", International Journal of Computational Intelligence Systems, vol. 5, no. 3, 2012, pp. 597–610, DOI 10.1080/18756891.2012.696923

[4] M. Abuteir and A. El-Halees, "Mining educational data to improve students' performance: A case study", International Journal of Information and Communication Technology Research, vol. 2, 01 2012, pp. 140–146

[5] M. Husain and S. Khan, "Students' feedback: An effective tool in teachers' evaluation system", International Journal of Applied and Basic Medical Research, vol. 6, 07 2016, p. 178, DOI 10.4103/2229-516X.186969

[6] H. Ambos, N. Tran, and A. Jung, "The logistic network lasso", CoRR, vol. abs/1805.02483, 2018

[7] A. Jung, "Learning networked exponential families with network lasso", CoRR, vol. abs/1905.09056, 2019

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space", 1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings, 2013

[9] T. Mikolov, W.-t. Yih, and G. Zweig, "Linguistic regularities in continuous space word representations", Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Atlanta, Georgia, June 2013, pp. 746–751

[10] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality", Advances in Neural Information Processing Systems, vol. 26, 10 2013

[11] Q. V. Le and T. Mikolov, "Distributed representations of sentences and documents", CoRR, vol. abs/1405.4053, 2014

[12] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation", CoRR, vol. abs/1607.05368, 2016

[13] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification", CoRR, vol. abs/1607.01759, 2016

[14] J. Goodman, "Classes for fast maximum entropy training", CoRR, vol. cs.CL/0108006, 2001

[15] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation", Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543

[16] R. Socher, C. C. Lin, A. Y. Ng, and C. D. Manning, "Parsing Natural Scenes and Natural Language with Recursive Neural Networks", Proceedings of the 26th International Conference on Machine Learning (ICML), 2011

[17] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council, vol. 5, 02 1994, pp. 157–66, DOI 10.1109/72.279181

[18] S. Hochreiter and J. Schmidhuber, "Long short-term memory", Neural computation, vol. 9, 12 1997, pp. 1735–80, DOI 10.1162/neco.1997.9.8.1735

[19] K. S. Tai, R. Socher, and C. D. Manning, "Improved semantic representations from tree-structured long short-term memory networks", Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), Beijing, China, July 2015, pp. 1556–1566, DOI 10.3115/v1/P15-1150

[20] A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with deep bidirectional lstm", 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, Dec 2013, pp. 273–278, DOI 10.1109/ASRU.2013.6707742

[21] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards universal paraphrastic sentence embeddings", 4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings, 2016

[22] R. Kiros, Y. Zhu, R. Salakhutdinov, R. S. Zemel, A. Torralba, R. Urtasun, and S. Fidler, "Skip-thought vectors", CoRR, vol. abs/1506.06726, 2015

[23] E. Pavlick, P. Rastogi, J. Ganitkevitch, B. Durme, and C. Callison-Burch, "Ppdb 2.0: Better paraphrase ranking, fine-grained entailment relations, word embeddings, and style classification", 01 2015, pp. 425–430, DOI 10.3115/v1/P15-2070

[24] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "From paraphrase database to compositional paraphrase model and back", Transactions of the Association for Computational Linguistics, vol. 3, 06 2015, DOI 10.1162/tacl_a_00143

[25] S. Arora, Y. Li, Y. Liang, T. Ma, and A. Risteski, "Random walks on context spaces: Towards an explanation of the mysteries of semantic word embeddings", CoRR, vol. abs/1502.03520, 2015

[26] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings", 5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24 - 26, 2017, Conference Track Proceedings, 2017

[27] M. Marelli, S. Menini, M. Baroni, L. Bentivogli, R. Bernardi, and R. Zamparelli, "The SICK (Sentences Involving Compositional Knowledge) dataset for relatedness and entailment", May 2014, DOI 10.5281/zenodo.2787612

[28] O. Kuchaiev and B. Ginsburg, "Factorization tricks for LSTM networks", CoRR, vol. abs/1703.10722, 2017

[29] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. V. Le, G. E. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer", CoRR, vol. abs/1701.06538, 2017

[30] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need", CoRR, vol. abs/1706.03762, 2017

[31] M. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation", CoRR, vol. abs/1508.04025, 2015

[32] P. J. Liu, M. Saleh, E. Pot, B. Goodrich, R. Sepassi, L. Kaiser, and N. Shazeer, "Generating wikipedia by summarizing long sequences", CoRR, vol. abs/1801.10198, 2018

[33] N. Kitaev and D. Klein, "Constituency parsing with a self-attentive encoder", CoRR, vol. abs/1805.01052, 2018

[34] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations", CoRR, vol. abs/1802.05365, 2018

[35] C. Chelba, T. Mikolov, M. Schuster, Q. Ge, T. Brants, and P. Koehn, "One billion word benchmark for measuring progress in statistical language modeling", CoRR, vol. abs/1312.3005, 2013

[36] M. E. Peters, W. Ammar, C. Bhagavatula, and R. Power, "Semi-supervised sequence tagging with bidirectional language models", CoRR, vol. abs/1705.00108, 2017

[37] B. McCann, J. Bradbury, C. Xiong, and R. Socher, "Learned in translation: Contextualized word vectors", CoRR, vol. abs/1708.00107, 2017

[38] H. Inan, K. Khosravi, and R. Socher, "Tying word vectors and word classifiers: A loss framework for language modeling", CoRR, vol. abs/1611.01462, 2016

[39] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization", CoRR, vol. abs/1607.06450, 2016

[40] Bowman, S. R., Angeli, Gabor, Potts, Christopher, Manning, and C. D., "A large annotated corpus for learning natural language inference", Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2015

[41] J. Howard and S. Ruder, "Fine-tuned language models for text classification", CoRR, vol. abs/1801.06146, 2018
[42] J. Blitzer, M. Dredze, and F. Pereira, "Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification.", 01 2007
[43] L. Mou, Z. Meng, R. Yan, G. Li, Y. Xu, L. Zhang, and Z. Jin, "How transferable are neural networks in NLP applications?", CoRR, vol. abs/1603.06111, 2016
[44] A. M. Dai and Q. V. Le, "Semi-supervised sequence learning", CoRR, vol. abs/1511.01432, 2015
[45] S. Merity, C. Xiong, J. Bradbury, and R. Socher, "Pointer sentinel mixture models", CoRR, vol. abs/1609.07843, 2016
[46] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson, "How transferable are features in deep neural networks?", CoRR, vol. abs/1411.1792, 2014
[47] L. N. Smith, "No more pesky learning rate guessing games", CoRR, vol. abs/1506.01186, 2015
[48] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift", CoRR, vol. abs/1502.03167, 2015
[49] B. Felbo, A. Mislove, A. Søgaard, I. Rahwan, and S. Lehmann, "Using millions of emoji occurrences to learn any-domain representations for detecting sentiment, emotion and sarcasm", 08 2017
[50] S. Merity, N. S. Keskar, and R. Socher, "Regularizing and optimizing LSTM language models", CoRR, vol. abs/1708.02182, 2017
[51] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving language understanding with unsupervised learning", Technical report, OpenAI, 2018
[52] Y. Bengio and Y. LeCun, eds., "4th international conference on learning representations, ICLR 2016, san juan, puerto rico, may 2-4, 2016, conference track proceedings", 2016
[53] Y. Zhu, R. Kiros, R. S. Zemel, R. Salakhutdinov, R. Urtasun, A. Torralba, and S. Fidler, "Aligning books and movies: Towards story-like visual explanations by watching movies and reading books", CoRR, vol. abs/1506.06724, 2015
[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings, 2015
[55] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding", CoRR, vol. abs/1810.04805, 2018
[56] T. Linzen, E. Dupoux, and Y. Goldberg, "Assessing the ability of lstms to learn syntax-sensitive dependencies", Transactions of the Association for Computational Linguistics, vol. 4, 11 2016, DOI 10.1162/tacl_a_00115
[57] K. Gulordava, P. Bojanowski, E. Grave, T. Linzen, and M. Baroni, "Colorless green recurrent networks dream hierarchically", CoRR, vol. abs/1803.11138, 2018
[58] A. Radford, R. Józefowicz, and I. Sutskever, "Learning to generate reviews and discovering sentiment", CoRR, vol. abs/1704.01444, 2017
[59] W. L. Taylor, "Cloze procedure: A new tool for measuring readability", Journalism Quarterly, vol. 30, no. 4, 1953, pp. 415–433, DOI 10.1177/107769905303000401
[60] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey, J. Klingner, A. Shah, M. Johnson, X. Liu, L. Kaiser, S. Gouws, Y. Kato, T. Kudo, H. Kazawa, K. Stevens, G. Kurian, N. Patil, W. Wang, C. Young, J. Smith, J. Riesa, A. Rudnick, O. Vinyals, G. Corrado, M. Hughes, and J. Dean, "Google's neural machine translation system: Bridging the gap between human and machine translation", CoRR, vol. abs/1609.08144, 2016
[61] M. Schuster and K. Nakajima, "Japanese and korean voice search", 03 2012, pp. 5149–5152, DOI 10.1109/ICASSP.2012.6289079
[62] P. Vincent, H. Larochelle, Y. Bengio, and P.-A. Manzagol, "Extracting and composing robust features with denoising autoencoders", 01 2008, pp. 1096–1103, DOI 10.1145/1390156.1390294
[63] Alex Jung, "CS-E3210 - Machine Learning: Basic Principles", 2018, https://mycourses.aalto.fi/course/view.php?id=20569
[64] S. Bird, E. Klein, and E. Loper, "Natural language processing with python", O'Reilly Media Inc., 2009, ISBN: 9780596516499 0596516495

[65] C. D. Manning, P. Raghavan, and H. Schütze, "Introduction to information retrieval", Cambridge University Press, 2008, ISBN: 0521865719, 9780521865715

[66] L. Wu, "Clustering text-formed course feedback data", Master Thesis at Aalto University, diplomityö, 2018-11-07

[67] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, "GLUE: A multi-task benchmark and analysis platform for natural language understanding", CoRR, vol. abs/1804.07461, 2018

[68] W. B. Dolan and C. Brockett, "Automatically constructing a corpus of sentential paraphrases", Proceedings of the Third International Workshop on Paraphrasing (IWP2005), 2005

[69] D. Cer, M. Diab, E. Agirre, I. Lopez-Gazpio, and L. Specia, "SemEval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation", Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017), Vancouver, Canada, August 2017, pp. 1–14, DOI 10.18653/v1/S17-2001

[70] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach", CoRR, vol. abs/1907.11692, 2019

[71] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "ALBERT: A Lite BERT for Self-supervised Learning of Language Representations", arXiv e-prints, Sep 2019, p. arXiv:1909.11942

[72] D. M. Hawkins, "The problem of overfitting", Journal of Chemical Information and Computer Sciences, vol. 44, no. 1, 2004, pp. 1–12, DOI 10.1021/ci0342472. PMID: 14741005

[73] 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, vol. 12, 2011, pp. 2825–2830

[74] D. J. KETCHEN and C. L. SHOOK, "The application of cluster analysis in strategic management research: An analysis and critique", Strategic Management Journal, vol. 17, no. 6, 1996, pp. 441–458, DOI 10.1002/(SICI)1097-0266(199606)17:6<441::AID-SMJ819>3.0.CO;2-G

[75] D. Sculley, "Web-scale k-means clustering", 01 2010, pp. 1177–1178, DOI 10.1145/1772690.1772862

[76] W. Hämäläinen, V. Kumpulainen, and M. Mozgovoy, "Evaluation of clustering methods for adaptive learning systems", Artificial Intelligence Applications in Distance Education, 2015, pp. 237–260

[77] B. J. Frey and D. Dueck, "Clustering by passing messages between data points", Science, vol. 315, no. 5814, 2007, pp. 972–976, DOI 10.1126/science.1136800

[78] R. Guan, X. Shi, M. Marchese, C. Yang, and Y. Liang, "Text clustering with seeds affinity propagation", IEEE Transactions on Knowledge and Data Engineering, vol. 23, April 2011, pp. 627–637, DOI 10.1109/TKDE.2010.144

[79] D. Dueck and B. Frey, "Non-metric affinity propagation for unsupervised image categorization", 11 2007, pp. 1–8, DOI 10.1109/ICCV.2007.4408853

[80] M. Leone, S. Sumedha, and M. Weigt, "Clustering by soft-constraint affinity propagation: Applications to gene-expression data", Bioinformatics (Oxford, England), vol. 23, 11 2007, pp. 2708–15, DOI 10.1093/bioinformatics/btm414

[81] Jianbo Shi and J. Malik, "Normalized cuts and image segmentation", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 22, Aug 2000, pp. 888–905, DOI 10.1109/34.868688

[82] A. Ng, M. Jordan, and Y. Weiss, "On spectral clustering: Analysis and an algorithm", Adv. Neural Inf. Process. Syst, vol. 14, 04 2002

[83] L. Zelnik-Manor and P. Perona, "Self-tuning spectral clustering", 01 2004

[84] T. Xia, J. Cao, Y.-d. Zhang, and J.-t. Li, "On defining affinity graph for spectral clustering through ranking on manifolds", Neurocomputing, vol. 72, 08 2009, pp. 3203–3211, DOI 10.1016/j.neucom.2009.03.012

[85] M. Lucińska and S. Wierzchon, "Spectral clustering based on k-nearest neighbor graph", 09 2012, DOI 10.1007/978-3-642-33260-9_22

[86] P. Perona and W. Freeman, "A factorization approach to grouping", Computer Vision — ECCV'98 (H. Burkhardt and B. Neumann, eds.), Berlin, Heidelberg, 1998, pp. 655–670
[87] G. L. Scott and H. C. Longuet-higgins, "Feature grouping by relocalisation of eigenvectors of proximity matrix", in Proceedings of British Machine Vision Conference, 1990, pp. 103–108
[88] T. Shi, M. Belkin, and B. Yu, "Data spectroscopy: Eigenspaces of convolution operators and clustering", The Annals of Statistics, vol. 37, 07 2008, DOI 10.1214/09-AOS700
[89] D. Cvetkovic, "Signless laplacian and line graph", Bulletin. Classe des Sciences Mathématiques et Naturelles. Sciences Mathématiques, vol. 131, 01 2005, DOI 10.2298/BMAT0530085C
[90] M. Newman, "Detecting community structure in networks", Eur Phys J, vol. 38, 03 2004
[91] M. Newman and M. Girvan, "Finding and evaluating community structure in networks", Physical review. E, Statistical, nonlinear, and soft matter physics, vol. 69, 03 2004, p. 026113, DOI 10.1103/PhysRevE.69.026113
[92] B. Schölkopf, A. Smola, and K.-R. Müller, "Nonlinear component analysis as a kernel eigenvalue problem", Neural Computation, vol. 10, 07 1998, pp. 1299–1319, DOI 10.1162/089976698300017467
[93] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "A density-based algorithm for discovering clusters in large spatial databases with noise", 1996, pp. 226–231
[94] A. K. Jain and R. C. Dubes, "Algorithms for clustering data", Prentice-Hall, Inc., 1988, ISBN: 013022278X
[95] "A comparison of internal and external cluster validation indexes", Stevens Point, Wisconsin, USA, World Scientific and Engineering Academy and Society (WSEAS), 2011
[96] Wikipedia contributors, "Confusion matrix — Wikipedia, the free encyclopedia." https://en.wikipedia.org/w/index.php?title=Confusion_matrix&oldid=935615638, 2020, [Online; accessed 29-January-2020]
[97] L. Hubert and P. Arabie, "Comparing partitions", Journal of Classification, vol. 2, Dec 1985, pp. 193–218, DOI 10.1007/BF01908075
[98] "Information theoretic measures for clusterings comparison: Is a correction for chance necessary?", New York, NY, USA, Association for Computing Machinery, 2009
[99] E. B. Fowlkes and C. L. Mallows, "A method for comparing two hierarchical clusterings", Journal of the American Statistical Association, vol. 78, no. 383, 1983, pp. 553–569, DOI 10.1080/01621459.1983.10478008
[100] A. Rosenberg and J. Hirschberg, "V-measure: A conditional entropy-based external cluster evaluation measure.", 01 2007, pp. 410–420
[101] H. Becker, "Identification and characterization of events in social media", doctoral thesis at columbia university, 2011
[102] P. Rousseeuw, "Rousseeuw, p.j.: Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. comput. appl. math. 20, 53-65", Journal of Computational and Applied Mathematics, vol. 20, 11 1987, pp. 53–65, DOI 10.1016/0377-0427(87)90125-7
[103] D. L. Davies and D. W. Bouldin, "A cluster separation measure", IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. PAMI-1, April 1979, pp. 224–227, DOI 10.1109/TPAMI.1979.4766909
[104] T. Caliński and H. JA, "A dendrite method for cluster analysis", Communications in Statistics - Theory and Methods, vol. 3, 01 1974, pp. 1–27, DOI 10.1080/03610927408827101
[105] D. Moulavi, P. A Jaskowiak, R. Campello, A. Zimek, and J. Sander, "Density-based clustering validation", 04 2014, DOI 10.1137/1.9781611973440.96
[106] D. Moulavi, P. A Jaskowiak, R. Campello, A. Zimek, and J. Sander, "Density-based clustering validation", 04 2014, DOI 10.1137/1.9781611973440.96
[107] V. Lempitsky, P. Kohli, C. Rother, and T. Sharp, "Image segmentation with a bounding box prior", 11 2009, pp. 277 – 284, DOI 10.1109/ICCV.2009.5459262
[108] M. E. J. Newman, "Networks: An introduction", Oxford Univ. Press, 2010, ISBN: 9780199206650
[109] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein, "Distributed optimization and statistical learning via the alternating direction method of multipliers", now, 2011, ISBN: null
[110] R. Řehůřek and P. Sojka, "Software Framework for Topic Modelling with Large Corpora", Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, Valletta,

Malta, May 2010, pp. 45–50. http://is.muni.cz/publication/884893/en

[111] Jason Brownlee Machine Learning Mastery, https://machinelearningmastery.com/tactics-to-combat-imbalanced-classes-in-your-machine-learning-dataset

[112] N. Tran, H. Ambos, and A. Jung, "Classifying partially labeled networked data via logistic network lasso", CoRR, vol. abs/1903.10926, 2019