Partially Automated Grading of Short Free-text Responses in Computer Science through Sentence Embedding and Clustering

---

Sheena Philip

Supervisor(s):
Prof. Richard Klein

A research report submitted in partial fulfillment of the requirements for the degree of Master of Science in the field of e-Science

in the

School of Computer Science and Applied Mathematics
University of the Witwatersrand, Johannesburg

12 June 2023

Declaration

I, Sheena Philip, declare that this report is my own, unaided work. It is being submitted for the degree of Master of Science in the field of e-Science at the University of the Witwatersrand, Johannesburg. It has not been submitted for any degree or examination at any other university.

Sheena Philip

12 June 2023

Abstract

A significant portion of educators' time is spent marking assessments, which could be better utilized for teaching and research to enhance the overall education experience. To assess higher-order thinking, questions that require short text answers are necessary. However, automatically grading these questions is much more complex since computers need to understand the underlying semantic meaning of the text. Furthermore, the dataset available for grading is limited to a few hundred responses due to the smaller size of lecture classes, which is not sufficient for evaluating most NLP and machine learning methods. To address this, this research aims to partially automate the grading of short free-text responses in computer science by grouping similar responses and manually marking specific submissions that best represent the group. It will explore various sentence embedding techniques, clustering techniques, and sampling techniques, and evaluate the Enhancement of Clustering by Iterative Classification (ECIC) algorithm, which improves cluster quality. The study found that Agglomerative clustering combined with Universal Sentence Encoder (USE) and a sampling strategy that marks submissions based on their distance to the center of the cluster produced the best results, balancing time saved and meeting the performance criteria. This combination resulted in a 65% reduction in the time it takes to grade a question. However, the ECIC algorithm was not effective on datasets that comprises a few hundred data points.

Acknowledgements

I express my gratitude to Prof. Richard Klein for his invaluable guidance throughout this undertaking. The path to reaching this point has been fraught with challenges and unforeseen circumstances, but his kindness and willingness to accommodate me is something I deeply appreciate. I thank God for His provision throughout my life. I extend my appreciation to my mother, whose unwavering support and encouragement have been the driving force behind my personal and professional growth. Her selfless devotion and inspiration have molded me into the woman I am today. I thank the rest of my family for their prayers and encouragement throughout this process. Finally, my thanks goes to DST-CSIR National e-Science Postgraduate Teaching and Training Platform for providing the financial support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the advance of AI, methods to improve on the education system are constantly being explored. Assessment and marking are tedious but essential components of an educators role. Mason and Grove-Stephensen [22] notes that 30% of British teachers time is spent on marking and 40% is spent in a classroom. A teachers time is nearly split equally between these two components even though it is noted that teaching is the most valued activity for a teacher [22].

University lecturers are also subjected to enormous work pressure due to the ever increasing demands in the tertiary education sector and limited resources available. Lecturers are made to balance lecturing, marking, research, supervising and administrative tasks. Professors in the U.S spend between 50-60 hours working per week which is approximately 50% more than the expected 40 hour week [17]. Therefore, being able to reduce the amount of time an educator spends on marking assessments can greatly contribute to the quality of education a student receives as well as the mental sanity of an educator, as the time saved can be reallocated to other areas of better use.

Bloom's Taxonomy is a hierarchical classification of human cognitive skills which can guide teachers when setting questions in order to ensure a student's understanding is comprehensively being tested. The automation of structured forms of assessment such as Multiple Choice questions, Mathematical questions and True/False questions has already been done due to their simplicity. However, according to Blooms Taxonomy these forms of assessment generally test lower order thinking and many researchers consider this to be an inadequate form of testing especially in relation to complex subjects. For this reason, paragraph and essay questions are

used more often to test higher order thinking [18].

This research aims to make steps in the automatic assessment of short-free text questions by developing an automatic short free-text marking model. The data set which will be used in this research comprises hundreds of student answers to a questions set in the Computer Science domain. The research does this by using methods in Natural Language Processing (NLP) to represent the text in a manner which embeds its meaning. Clustering is then performed to group similar answers and finally a sampling strategy is implemented to determine which papers should be marked by an educator to best represent the cluster. The average of the papers marked or an alternative methods are then applied to the entire cluster thereby significantly reducing the number of papers that an educator is required to mark.

The remainder of this document is organised as follows. Chapter 2 presents a literature review of existing methods used both in terms of automated marking systems as well as with short text clustering approaches. Chapter 3 gives relevant background information that is key to understanding sentence embeddings and clustering. Chapter 4 describes the problem statement, research questions, research aim and objectives as well as the research methodology that will be followed. In Chapter 5, the results are presented, and a comprehensive discussion of these results is provided. Finally, concluding remarks are presented in Chapter 6.

Chapter 2

Related work

2.1 Automated marking systems

Knowledge-based marking systems is one method for automating the evaluation and grading of student responses. These systems rely on domain-specific knowledge and rule-based algorithms to assess the accuracy and quality of student answers. Siddiqi, Harrison, and Siddiqi [31] describes a system called IndusMarking. The system is specifically designed to evaluate factual answers that have a definitive criterion for correctness.

It utilizes a structure matching approach, which involves comparing the content of the student's answer text with a pre-defined structure created using a dedicated structure editor. This matching process allows the system to assess the alignment between the expected structure and the student's response, determining the accuracy of the answer based on this comparison.

The findings reveal that these systems offer potential benefits in terms of efficiency and timely feedback. However, the study also identified limitations, such as the systems' difficulty in capturing complex or nuanced responses that require higher-order thinking skills. General problems with knowledge-based marking systems is the need for robust knowledge representation and rule development to accommodate diverse assessment tasks.

Model based grading systems use machine learning to try overcome the problems associated with knowledge-based marking systems. Mohler and Mihalcea [25] use

unsupervised techniques to automatically mark short text answers. They investigate a number of corpus and knowledge based measures of similarity whilst also introducing a technique which automatically integrates feedback from student answers. The results were compared to the results from tf-idf, the baseline similarity measure at the time. The model was successful as it's performance exceeded the baseline Pearson correlation of 0.3647.

Dong, Zhang, and Yang [11] uses an attention-based recurrent convolutional neural network to grade essays automatically. The model treats essays as sentence-document hierarchies and uses attention pooling to automatically find the relevant words and sentences and determine their relative weights. When evaluated, their system had a 0.764 average Quadratic Weighted Kappa (QWK) value when only using word features and a QWK value of 0.738 when only using character features. The model outperformed all models at the time which they chose to investigate.

2.2 Short text clustering

The limited number of words within each text results in a sparseness of the associated vector representation. This is a challenge which various clustering methods in literature have tried to overcome.

Xu et al. [34] proposes a model termed STC^2. Firstly, Raw text is embedded into binary codes using unsupervised dimensionality reduction methods. Then word embeddings, created using word2vec, are inputted into a convolutional neural network to learn deep feature representations of the text. These are then combined and clustering is performed. Another method, proposed by Hadifar et al. [14], uses an autoencoder and weighted word embedding to learn discriminative features and then uses allocations from a clustering algorithm to supervise and update the weights of the encoder network. The method is found to be effective on three short text datasets.

Biterm topic modelling(BTM), proposed by Cheng et al. [7], is a topic modelling approach that identifies topics from word co-occurence patterns (biterms). Clustering

is then used to allocate a piece of text to its most probable topic. BTM is able to identify more distinct and meaningful topics .

A text augmentation method, as proposed by Banerjee, Ramanathan, and Gupta [2], uses external sources such as Wikipedia to enhance the vector representation of short text. The short text is used to form a query and the returned relevant article titles are used as additional features to the short text. An issue with using external resources is that it can lead to a level of inconsistency.

A recent method, proposed by Zheng, Liu, and San Wong [36], does not use external sources but instead uses a corpus-based approach by using topic diffusion to find new words, which may not appear in the original short text, but relate to the content. The short text is used to determine possible topics for the relevant text. Additional words are then added by finding words related to the the topic based on the posterior probabilities between the new words given the original short text. Not using external sources but rather the dataset itself, ensures that the correctness of the enhancement is guaranteed given that it based on the exact same semantic structure of the entire dataset.

Yin and Wang [35] propose a short text clustering method which uses a collapsed Gibbs Sampling algorithm for the Dirichlet Process Mixture model, also known as GSDPMM. The number of clusters do not need to be specified in the beginning and the algorithm works by evaluating each text individually and assigning it to an existing cluster or a new cluster based on the probability of a document choosing a cluster with an evaluation metric similar to the Naive Bayes Classifier. More specifically, GSDPMM measures the similarity between a document and a cluster by evaluating the frequency of words that appear in a document and also in each cluster. This method effectively addresses the high dimensional problem of text clustering as GSDPMM only evaluates words in the current document and is therefore not influenced by other words in the vocabulary. Therefore, it is found that it has a low space and time complexity and scales effectively with large datasets.

Clustering serves as a fundamental component in modern topic modelling techniques. Leading examples of these methods, such as BertTopic [13] and Top2Vec

[1], employ a combination of techniques including UMAP and HDBSCAN to extract underlying topics or themes from a corpus of text data. While BertTopic incorporates BERT as the default sentence embedding prior to clustering, Top2Vec uses Doc2Vec to embed sentences before clustering. It is important to note that both techniques have been extended to include a variety of other sentence embeddings. In particular, BertTopic and Top2Vec have proven to be effective for short text data.

Another method, proposed by Rakib et al. [28], is based on enhancing existing clustering methods. The Data is embedded using 2 techniques namely: Glove which was trained on Wikipedia texts and BioASQ which uses the Word2Vec approach. Clustering is then performed using k-means, k-means– and hierarchical clustering in order to label the data. The proposed algorithm Enhancement of Clustering by Iterative Classification (ECIC) is then performed on the existing clusters.

The algorithm works by converting the text in each cluster to a tf-idf vector representation. It then identifies outliers, using an algorithm called Isolation Forest, in each cluster and reallocates the outliers to better suited clusters. This process is repeated numerous times until a stopping criteria is met. In each iteration a training set which comprises of non-outliers and their cluster labels and a test set which comprises of outliers are formed. A classifier (Multinomial Logistic Regression) is then trained using the training data set and the test set is classified.

The experimental results show that ECIC improves the cluster quality of all the different baseline clustering methods and outperforms other short text clustering methods, specifically methods described by Xu et al. [34] and Hadifar et al. [14] with a statistically significant margin. Pugachev and Burtsev [27] improves on ECIC by using modern sentence embeddings methods namely USE [6], BERT [29] and RoBERTa [20]. In both papers, experiments were conducted using datasets which are in the range of thousands of datapoints. This research aims to use these methods as a foundation and investigate how they perform on a much smaller dataset.

Chapter 3

Background

3.1 Transfer Learning

NLP models that are built from scratch require huge amounts of data in order to train the neural networks to produce a reasonable accuracy. However, this is not always feasible given that the data is not readily available and the computing power required to process such large amounts of information is enormous. In some cases, the models needs thousands of hours of training time. To close this gap, models were trained on extremely large corpura, resulting in pre-trained models that were then released to the public to fine tune for specific purposes.

It was discovered that deep networks learn in a hierarchical manner where the lower layers represented simple/generalised features and the higher layers represented more complex/specialised features. Instead of training a new neural network from scratch each time, the lower layers of a neural network could be transferred for use in another application. As a result, the only part that had to be trained was the top layers which significantly reduce the training time and resources required and can drastically improve results [9, 6, 10].

3.2 Sentence Embeddings

Word embeddings are learned numerical vector representation of words that capture the context as well as the semantic and syntactic similarity of words in relation

to other words in a document. Word embeddings on their own do not however capture the meaning of whole sentences, this is that the context of sentences can vary greatly even if the words in the sentence are very similar. To overcome this problem sentence embedding techniques were created in order to represent the whole sentence as a vector whilst ensuring that the sentence's semantic information in relation to other sentences is maintained [9, 6, 10].

3.2.1 USE

The Universal Sentence Encoder (USE) was published in 2018. The USE model is trained and optimized for a variety of various sized texts, such as sentences, phrases or short paragraphs [6]. It is trained on a diverse range of data sources and tasks with the aim of accommodating, through the concept of transfer learning, a wide variety of NLP tasks.

Cer et al. [6] explain the USE model encodes variable length English text and produces as an output of a fixed 512 dimensional vector embedding representation of each piece of text.

There are 2 types of encoders within the USE encoder family. The first encoder model uses a transformer architecture aiming for high accuracy at the expense of resource usage and greater model complexity. The transformer uses attention to create context based representations of words which takes into consideration the identity of the other words as well as the ordering. The representation of the words is then converted into a fixed encoding by calculating the element wise sum of the representations at each word position [6].

The second encoder model relies on the Deep Averaging Network (DAN) architecture that has slightly reduced accuracy but is less resource demanding. The inputted word embeddings are averaged together and then fed into a feedforward deep neural network(DNN) to form fixed length sentence embeddings. Thought

must be given to which USE encoder model to use given time and resource constraints [6].

3.2.2 BERT and SentenceBert

Bidirectional Encoder Representations from Transformers (BERT) proposed by Devlin et al. [10] is another embedding widely used in industry since Google open sourced it in 2018. It also utilises the concept of transfer learning. Bert is pretrained on an extremely large corpus of unlabelled text which includes the whole of Wikipedia and the Book Corpus. The unsupervised model can then be fined tuned for specific NLP tasks.

Bert makes uses of a Transformer which is an attention method that learns contextual relationships between words in a piece of text. Transformers normally comprise of an encoder and a decoder. However, BERT only uses the encoder mechanism as the goal is to learn embeddings. Some models read the text either from right to left or left to right in order to train the model and determine context. However, BERT reads the whole sequence of words at one time and is therefore a truly bidirectional model.

Each text is represented as a set of tokens, where specific tokens are used for spaces, separator [SEP], mask [MASK], punctuation etc. Other language models use prediction as the learning goal. They predict the word which will follow a set of words. However, this in itself is a directional approach which limits context learning. Bert uses the two following learning strategies to overcome this:

1. Masked LM (MLM):15% of the words in each sequence are replaced with a [MASK] token before they are fed into BERT. The model then tries to predict the masked words based on the context of the surrounding non-masked words. The BERT loss function only considers the prediction of the masked values and not the non-masked words. This results in the model converging slower that other directional models but results in greater context awareness.

2. Next Sentence Prediction (NSP): The model is trained by receiving pairs of sentences. In 50% of the cases the pairs is correct (the second sentence originally follows the first sentence) and in the remaining 50% of the cases the sentences are not correct (the second sentence is not related to the first). To allow the model to distinguish between the two sentences, a [CLS] token is placed at the start of the first sentence and a [SEP] token is placed inbetween the two sentences.A positional embedding is also added to each token. To predict whether the two sentences are related or not, BERT inputs the entire sequence into the Transformer model and calculates the probability that the two are related using a softmax function.

After training is complete, BERT inputs a sentence vector after it has been tokenized and forms a resulting vector for each word in the text. This format is not suitable for sentence embedding as we need a single vector to represent the text. To overcome this, a sentence embedding can be created by simply averaging the word embeddings or by using the [CLS] token which contains the classification of the entire text. This can be therefore be used as the text embedding.

The sentence embeddings produced by these methods did not produce high quality embeddings. SentenceBERT proposed by Reimers and Gurevych [29] aimed to solve this problem by using a Siamese Neural Network as seen in Figure 3.1 The network is trained using the SNLI dataset which is a collection of 570 000 sentence pairs with the labels that indicate whether the sentences are semantically the same, whether they contradict each other or whether they are neutral.

To generate semantically meaningful sentence embeddings using SBERT, the input text is first preprocessed and tokenized. Each sentence is then fed into a BERT transformer. A pooling operation is then performed to create a fixed sized sentence embedding. The pooling operation that is commonly used is computing the mean of all the word embeddings which is shown to generally have the highest performance.

The sentence embeddings u and v are then concatenated with the element-wise difference and multiplied by a trainable weight matrix $W \in \mathbb{R}^{3N \times K}$, where N is the dimension of the sentence embedding and K is the number of labels. This is then

passed through a softmax classifier which optimizes the cross-entropy loss. Upon inference, the two embeddings are compared using a cosine similarity function. Given that the research conducted in this paper only focuses on sentence embeddings BERT, will refer to SentenceBERT.

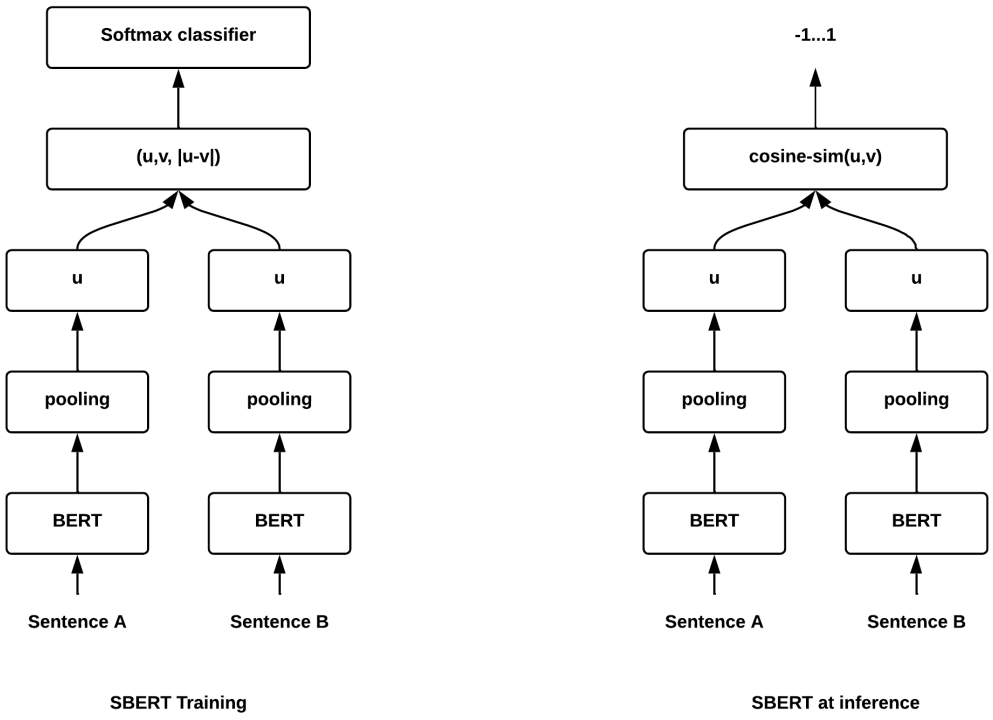


FIGURE 3.1: SBERT

3.2.3 Word2Vec and Doc2Vec

Word2vec as proposed by Mikolov et al. [24] is a powerful technique used in natural language processing (NLP) for representing words as vectors in a high-dimensional space. It uses a neural network to learn the vector representations of words by predicting the words that occur in the context of a given word. There are two main

algorithms used for training word2vec: Continuous Bag-of-Words (CBOW) and Skip-Gram as can be see in figure 3.2.

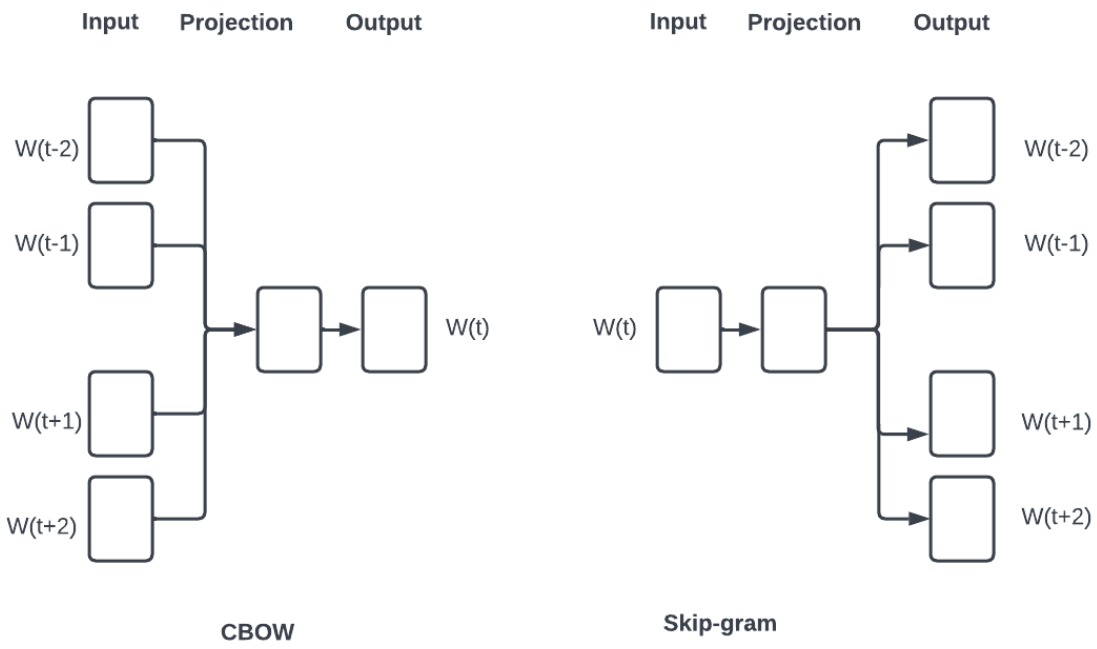


FIGURE 3.2: An overview of Word2Vec

In the CBOW algorithm, the neural network is trained to predict the target word given a set of context words. The context words are represented as one-hot vectors which are fed into an embedding layer initialized with random weights. The resulting word embeddings are then averaged out in a lambda layer. These embeddings are subsequently forwarded to a dense SoftMax layer that predicts the target word. By comparing the predicted target word with the actual one, the loss is calculated. During each epoch, the embedding layer is updated through backpropagation to optimize the loss.

In the Skip-Gram algorithm, the neural network is trained to predict the context words given a target word. To enable the skip-gram model to predict multiple words from a single given word, a pair of (A, B) is fed into the model, where A represents the input and B represents the label. This involves creating both positive

and negative input samples.

Positive input samples are structured as follows: [(target, context),1], where the target is the target word, the context represents the surrounding context words, and the label 1 indicates that the pair is relevant. Negative input samples are structured the same way: [(target, random),0], where the label 0 indicates that it is an irrelevant pair. However, instead of the actual surrounding words, randomly selected words are used in conjunction with the target words.

A overview of how the model works can be seen in figure 3.3. To generate dense word embeddings for both the target and context words, each pair is fed into individual embedding layers. The resulting embeddings are then multiplied using a "merge layer" to produce a dot product value. Next, the dot product value is passed through a dense sigmoid layer, which generates an output of either 0 or 1. The output is then compared to the actual label, and the loss is calculated. In each epoch, backpropagation is performed to update the embedding layer during the training process.

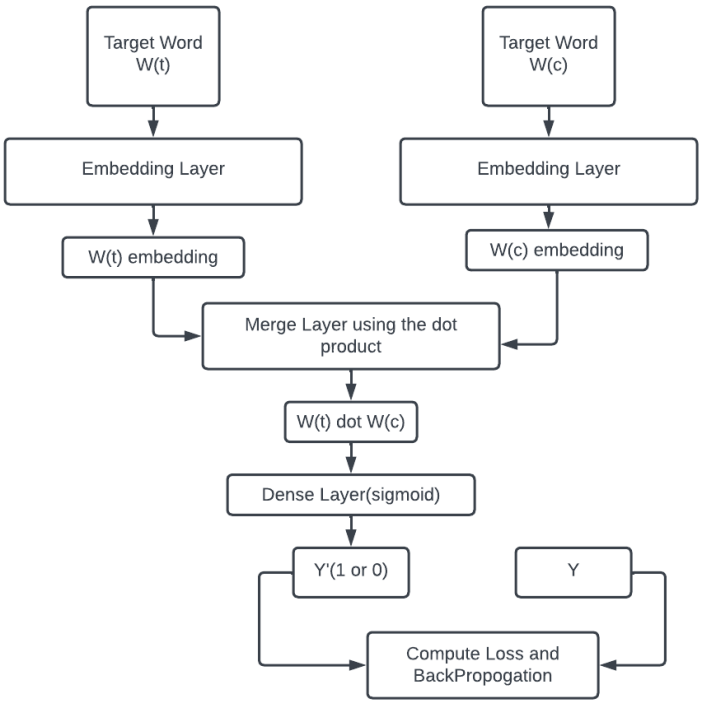


FIGURE 3.3: Skip-Gram algorithm in Word2Vec [30]

Doc2Vec, as proposed by Le and Mikolov [19], is an unsupervised algorithm derived from the popular technique Word2Vec but introduces an additional paragraph vector. There are two ways which this paragraph vector can be added to the model. The first, namely Distributed Memory version of Paragraph Vector (PV-DM) assigns a paragraph vector whilst still distributing word vectors amongst all sentences. The average of the paragraph vector and the word vectors is then taken in order to form the sentence embedding. This method is an extension of the continuous Bag-of-Words model (CBOW) only now the next sentence is predicted given a set of sentences.

The second, namely Distributed Bag of Words version of Paragraph Vector (PV-DOBW) is an extension of the Skip-gram algorithm. This algorithm randomly samples words from a sentence and then makes the model predict which sentence it came from [19].

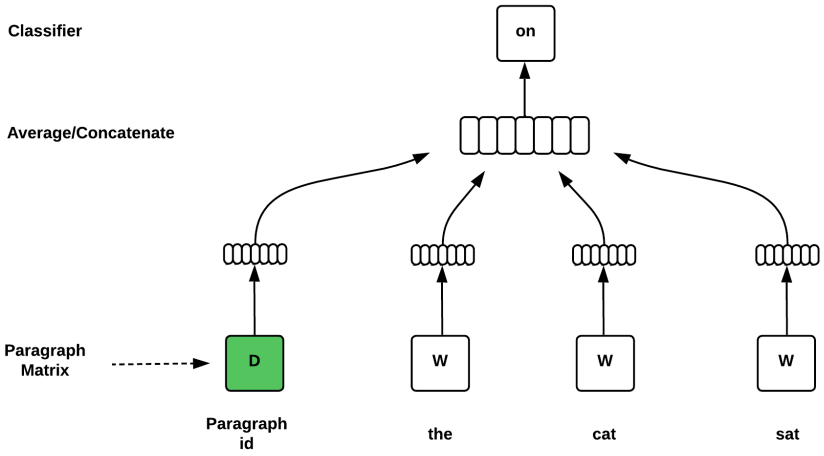


FIGURE 3.4: PV-DM model

3.3 Clustering techniques

3.3.1 Hierarchical clustering

There are 2 categories that hierarchical clustering [4] falls into namely the top-down (divisive) or bottom-up (agglomerative) approach. The agglomerative approach initially considers each data point as its own cluster and consecutively merges clusters which are similar until a single cluster or k clusters is formed. An example of this can be found in Figure 3.6.

In the first step the proximity matrix is calculated which is the distance between every single point given that each point is considered its own cluster at the start. In step two, similar clusters are merged together. In the example consider that data point b and c are similar and are merged into one cluster and data point d and e are similar and are merged into one cluster. In step 3 the proximity matrix is again calculated. The clusters def and bc are similar and are merged together to form a

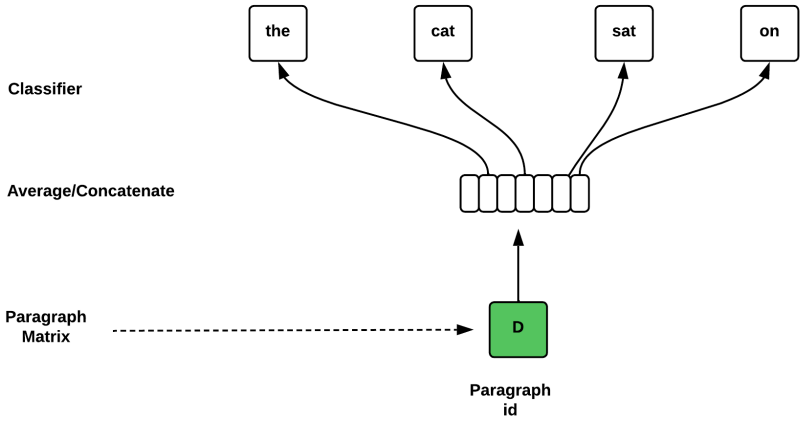


FIGURE 3.5: PV-DBOW model

new cluster. The resulting two clusters are a and bcdef. In the final step everything is merged together to form a single cluster.

A dendrogram, which is a tree like diagram, allows the sequence of merges to be visualised which can assist in determining the opitimal number of clusters to form. The divisive approach initially evaluates all points as one large cluster and with each iteration separates them into groups that are similar.

The distance between two clusters is typically defined using a distance metric such as Euclidean distance, Manhattan distance, or cosine distance. Let $d(i, j)$ be the distance between data points i and j, and let C and D be two clusters. Then, the distance between clusters C and D, denoted by $d(C, D)$, can be computed using different linkage criteria namely Single linkage, Complete linkage, Average linkage and Ward's method. Single linkage can be denoted by the following equation:

$$d(C, D) = \min d(i, j) : i \in C, j \in D \tag{3.1}$$

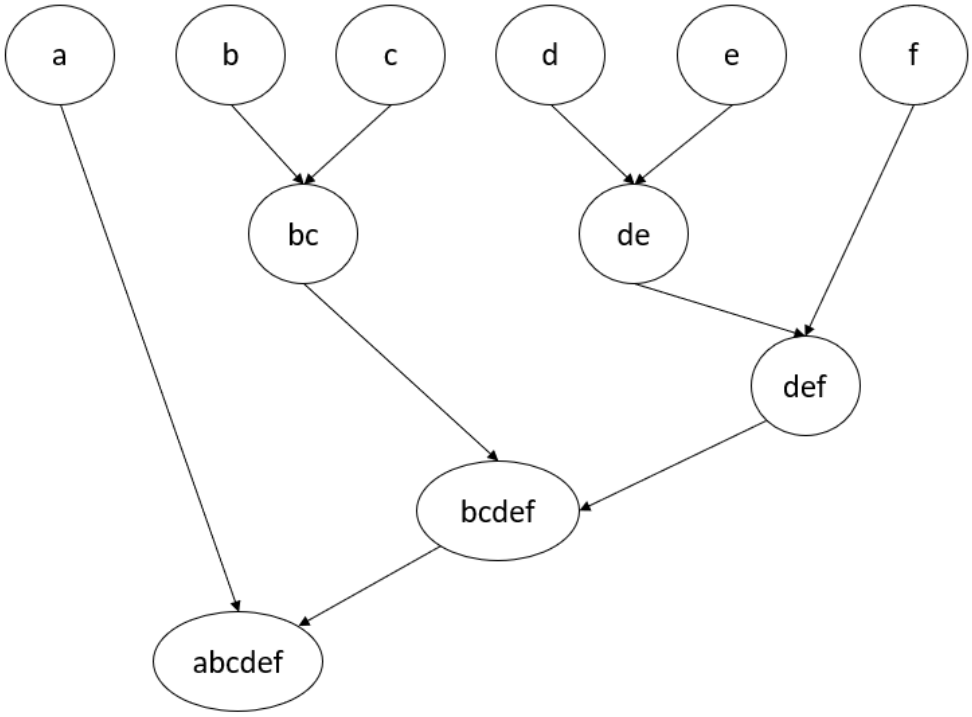


FIGURE 3.6: PV-DBOW model

It is good at separating non-elliptical shapes so long as the gap between clusters is not small. It does not perform well when there is noise between clusters. Complete linkage can be denoted by the following equation:

$$d(C, D) = \max d(i, j) : i \in C, j \in D \tag{3.2}$$

It is good at separating clusters if there is noise between clusters. However, it is biased towards globular clusters and it generally breaks up large clusters. Average linkage and Ward's linkage can be denoted by equation 2.3 and 2.4 respectively. They are both good at separating clusters if there is noise between them and are both biased towards globular clusters

$$d(C, D) = \frac{1}{|C||D|} \sum_{i \in C, j \in D} d(i, j) \tag{3.3}$$

$$d(C, D) = \frac{1}{|C||D|} \sum_{i \in C, j \in D} d(i, j)^2 \tag{3.4}$$

Centroid linkage can be denoted by the following equation and is not used very commonly:

$$d(C, D) = ||\mu(C) - \mu(D)||^2 \tag{3.5}$$

where $\mu(C)$ and $\mu(D)$ are the means of clusters C and D, respectively. The space complexity of hierarchical clustering is O(n^2) and the time complexity is O(n^3) which results in it not being an effective clustering method for large datasets. The divisive approach works in the same way, only this time all the data points are initially seen as a single cluster and then upon each iteration broken down into smaller clusters until each data point is its own cluster.

3.3.2 Gaussian mixed models

Gaussian mixture models (GMMs) [16] are popular method for clustering data. GMMs assume that the data is generated from a mixture of Gaussian distributions, each representing a cluster. The clusters are characterized by their mean and covariance matrix. GMM's are probabilistic models and use the soft clustering approach to assign data points to different clusters. GMM can be broken into three steps. In the first step the mean, covariance and weight parameters are randomly initialised. Every distribution is multiplied by a weight, given that there won't be equal number of samples for each distribution.

The next step is referred to as the Expectation step. It involves taking every data point and calculating the probability that it belongs to each distribution using the below equation:

$$r_{ic} = \frac{\pi_c \mathcal{N}(x_i|\mu_c, \Sigma_c)}{\sum_{j=1}^{k} \pi_j \mathcal{N}(x_i|\mu_j, \Sigma_j)}. \tag{3.6}$$

where k is the number of mixture components and π_i is the weight of the ith component, subject to $\pi_i > 0$ and $\sum_{i=1}^{k} \pi_i = 1$. This was initialised in the previous step. $\mathcal{N}(x|\mu_i, \Sigma_i)$ describes the probability density function (PDF) of a Gaussian distribution with covariance Σ and mean μ regarding each component as can be seen below:

$$\mathcal{N}(x_i|\mu_j, \Sigma_j) = \frac{1}{\sqrt{(2\pi)^k|\Sigma_c|}} exp(-\frac{1}{2}(x_i - \mu_c)^T\Sigma^{-1}(x_i - \mu_c)) \tag{3.7}$$

The final step refers to the maximization step where the various parameters for each distribution are updated based on the probabilities calculated previously. The step updates the parameters as follows:

$$\pi_c = \frac{\sum_{i=1}^{m} r_{ic}}{n} \tag{3.8}$$

$$\mu_c = \frac{\sum_{i=1}^{m} r_{ic} x_i}{\sum_{i=1}^{m} r_{ic}} \tag{3.9}$$

$$\Sigma_c = \frac{\sum_{i=1}^{m} r_{ic}(x_i - \mu_c)^2}{\sum_{i=1}^{m} r_{ic}} \tag{3.10}$$

where n refers to total number of datapoints. The probabilities are then calculated again with the updated distributions and the various parameters are updated again. This process continues till there is convergence.

3.3.3 UMAP

Uniform Manifold Approximation and Projection (UMAP) is a popular dimensionality reduction technique used in machine learning and data analysis. It was introduced in 2018 by McInnes, Healy, and Melville [23] and is based on the concept of constructing a low-dimensional representation of high-dimensional data that preserves its local structure. UMAP uses a combination of mathematical techniques, including stochastic gradient descent and Riemannian geometry, to minimize the cross-entropy between the high-dimensional data and its low-dimensional representation. The UMAP algorithm can be represented mathematically as follows:

Given a high-dimensional dataset X with n observations and p features, UMAP seeks to find a low-dimensional representation Y with m dimensions (where $m << p$) that preserves the pairwise distances between the observations in X. This is achieved by minimizing the following objective function:

$$E(Y) = \sum_{i,j} w_{i,j} d_{i,j}^2 (f_{i,j} - g_{i,j})^2 \tag{3.11}$$

where $d_{i,j}$ is the distance between the i-th and j-th observations in X, $f_{i,j}$ is the similarity between the observations in the high-dimensional space, and $g_{i,j}$ is the similarity between the observations in the low-dimensional space. The weights $w_{i,j}$ are used to give more importance to certain pairs of observations and are typically chosen based on the distance distribution of the data. The optimization problem is solved using stochastic gradient descent, with the gradients of the objective function computed using Riemannian geometry. Overall, UMAP provides a powerful tool for visualizing and analyzing high-dimensional data in a low-dimensional space.

3.3.4 HDBSCAN

HDBSCAN is a density-based clustering algorithm that was developed by Campello, Moulavi and Sander [5]. It is built upon the DBSCAN algorithm, which unlike DBSCAN, is able to identify clusters of various densities. There are five main steps in extracting a cluster. The first step is to estimate density, which is defined by computing the distance to the kth nearest neighbor. This is referred to as the Core Distance which will be represented as $core_k(x)$. A method is then required to split high density areas from low density areas. This is done by defining a new distance metric referred to as mutual reachability distance as follows:

$$d_{mreach-k}(a, b) = \max\{ \text{ core }_k(a),\ \text{ core }_k(b), d(a, b)\} \tag{3.12}$$

where $d(a, b)$ is the original metric distance between a and b. Using this metric, dense points which have a low core distance remain the same distance from one another and areas with sparse points are pushed away to ensure that they are atleast one core distance away from any other point.

The mutual reachability distance is utilized in the subsequent phase to create a minimum spanning tree, which aids in identifying densely connected points. The tree is built using Prim's algorithm which is constructed incrementally, by adding the edge with the minimum weight that connects the current tree to an unexplored

vertex at each step. The following step involves using the minimal spanning tree and converting it into a hierarchy of connected components. To accomplish this, it is convenient to sort the tree edges by distance in ascending order and traverse through them in reverse order. At each iteration a new cluster is formed by joining two clusters using a union-find data structure.

The next step involves condensing the large and normally complicated cluster hierarchy into a significantly smaller tree. This is done by using the minimum cluster size and iterating through the hierarchy and at each split determining whether each cluster has less points than the minimum cluster size. If this is determined to be true, the larger cluster absorbs the smaller cluster but takes note of which points were absorbed and the distance at which that happened. The minimum cluster size is the only parameter specified by the user which makes HDBSCAN significantly easier to use that DBSCAN, which has more parameters that are not as intuitive.

The final step involves extracting the clusters that have a longer lifetime or can be seen as being more stable. The metric that is used to calculate this is $\lambda = \frac{1}{distance}$. For a cluster, λ_{birth} is defined as when a cluster was created and λ_{death} is when the cluster partitioned into smaller clusters. Therefore, for a specific cluster, for every point p within it, λ_p is defined as the lambda value where the point left the cluster, either because it "fell out" of the cluster at some point or left the cluster when the cluster partitioned. It is a value that falls between λ_{birth} and λ_{death}. Therefore, stability can be defined as:

$$\sum_{p \in \text{cluster}} = (\lambda_p - \lambda_{\text{birth}}) \tag{3.13}$$

Every leaf node in the hierarchy is defined as a cluster. The tree is then traversed from the bottom to the top. If the combined stability of the child clusters is higher than the stability of the parent cluster, the stability of the parent cluster is defined to be the sum of its child clusters' stabilities. However, if the stability of the parent cluster is higher than the sum of its child clusters, the parent cluster is considered the selected cluster and all its child clusters are deselected. The current selected clusters are defined as the final clusters which results in flat clustering.

3.3.5 K-means

K-means is an iterative clustering algorithm [21]. Firstly the number of desired clusters/groups is specified and the corresponding number of data points are randomly initialised as the starting centroids. Each data point is then assigned to a cluster according to the minimum distance between the point and the cetroid. The mean is then calculated for each cluster and a new centroid is established. This process is repeated numerous times until the centroids do not change significantly or a certain number of epochs have been completed.

K-means has a linear complexity O(n) and is therefore relatively fast and scales well with larger datasets. However, the the number of clusters has to be specified which can be problematic in situations where this is not easily known. In addition, the initial centroids being randomly initialised yields results which are not necessarily repeatable or consistent.

Chapter 4

Research Methodology

The Research Methodology chapter serves as a comprehensive guide to understanding the systematic approach employed in this study. This chapter encompasses various sections that are vital for conducting rigorous research. Firstly, the problem statement concisely presents the specific issue or gap in knowledge that this research aims to address. Building upon the problem statement, the research question is formulated to precisely focus the investigation. The aim and objectives of the study further outline the broader objective and specific goals to be achieved.

Additionally, it is important to acknowledge the limitations of the study, which provide insights into the potential constraints and boundaries of the research. Lastly, the methodology section elucidates the detailed procedures, techniques, and tools employed to gather and analyze data, ensuring the reliability and validity of the research outcomes.

4.1 Problem statement

Educators across the world have been overloaded with work and are under significant time pressures due to a lack of resources. Educators should not have to spend any time on marking assignment questions but rather should be able to focus their limited time on teaching, research and other activities which better enhance the education experience. It is estimated that teachers spend 30% of their time on marking [22]. Automated systems have effectively taken over the grading process for Multiple Choice, Mathematical, and True/False questions.

However, these methods, as per Blooms Taxonomy, primarily target lower-order thinking skills and face difficulties when it comes to evaluating higher-order thinking abilities. For assessing higher-order thinking, the inclusion of paragraph and essay style questions becomes necessary. This however has not been fully automated with a 100% success rate and therefore there is room for improvement. In addition, the data sets which comprises of answers to an assignment question are in the range of only hundreds of data points which in itself is a challenge due to limited training data.

This problem can also be viewed as a NLP short text clustering problem whereby similar answers can be grouped together. The major challenge with this approach is the sparseness of the text's associated vector representation which results in it being difficult to cluster. Various methods have been implemented to overcome this which include methods based on text augmentation [2, 36], neural networks[34, 14], topic modelling [7] and the Dirichlet mixture model [35].

One of the most successful methods proposed by [28] uses the ECIC algorithm, as described above, to improve on existing clustering methods. However, the data sets used to test the algorithm are in the range of thousands of data points. It is unknown whether ECIC would work effectively on a data set which comprises of only a few hundred samples. This research explores the combination of different sentence embedding techniques, clustering techniques as well as sampling techniques in combination with ECIC [28] to create a system which can reduce the amount of time a lecturer spends on marking short free-text questions.

4.2 Research Questions

- What sentence embedding technique demonstrates the highest effectiveness in capturing the underlying meaning of the text, as evaluated through Pearsons Correlation and Cohen's Kappa?
- Which clustering technique proves to be the most effective in grouping similar responses, as assessed through Pearson's Correlation, Cohen's Kappa, Normalized Mutual Information, and Adjusted Rand Index?

- Which sampling technique exhibits the highest effectiveness in reducing the number of questions that necessitate manual marking, while accurately representing the cluster, as evaluated based on the percentage of documents requiring manual marking to achieve the required range of Cohen's Kappa and Pearson correlation?
- Is ECIC effective on a signifcantly smaller data set that comprises of only hundreds of data points?
- How much time does a lecturer save in marking?

4.3 Research Aims and Objectives

4.3.1 Research Aim

The aim of this research is to develop a system, using sentence embedding, clustering, ECIC and sampling methods, which will automatically mark short text questions in Computer Science, thereby reducing the manual marking time of a lecturer.

4.3.2 Objectives

The objectives of the research are:

- To embed the text using sentence embeding methods: USE, BERT and Doc2Vec.
- To apply the clustering methods K-means, Hierarchical agglomerative clustering, UMAP coupled with HDBSCAN, and GMM to group similar student responses.
- To implement the ECIC algorithm ontop of various clustering methods.
- To apply various sampling methods to determine which answers within each cluster to mark.
- To evaluate how the system performs and identify which combination of methods is most appropriate.

4.4 Limitations

The limitations of this research is that it will be evaluated only using Computer Science questions and therefore may not be transferable to other domains. In addition, short paragraphs will be only be used to train the models and therefore it may not be effective or longer pieces of text.

4.5 Assumptions

For the purposes of this research, it will be assumed that all the answers used to train the model are in the language English.

4.6 Ethics clearance

In order to utilize student answers for this study, ethical clearance has been obtained as it involves human participants. The study has received ethics clearance under the reference number CSAM-2021-01.

4.7 Methodology

The Flow diagram below represents an overview of the system that was created. Firstly data was acquired and pre-processed into a more suitable format. Each response was then vectorised and clustering was performed to group similar responses. ECIC was then applied to hopefully improve on clustering methods already performed. Finally a sampling strategy was implemented to determine which answers would be manually marked by the educator. The aim was to minimise how many answers need to be manually marked, while maintaining a level of accuracy and confidence in the automatically awarded grades.

4.7.1 Data Acquisition and prepossessing

The dataset consists of short paragraphs of text, which are responses from first-year Computer Science University students to questions posed by a lecturer, along with their corresponding grade. For each question, the dataset contains a few hundred

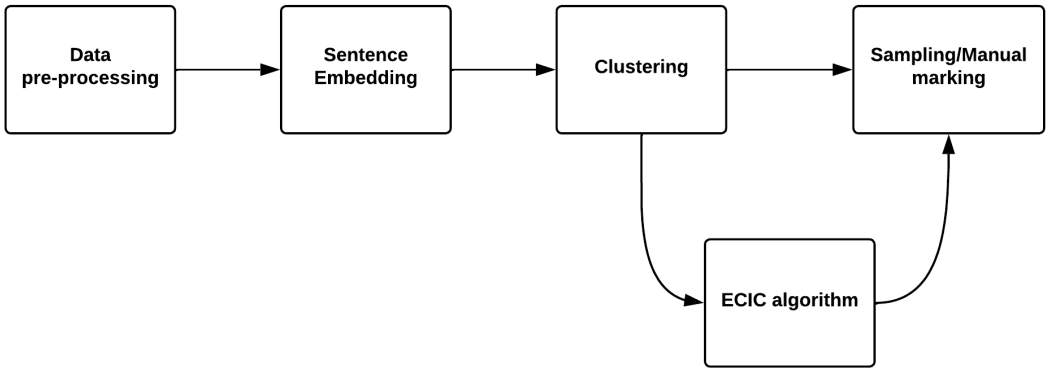


FIGURE 4.1: System Overview

student responses, ensuring that the system will be effective on a real-size class. Throughout this document, we will refer to the four datasets used in the investigation as Trees1, Queues2, Linked1, and Stacks3. A breakdown of each dataset can be seen in Table 4.1.

TABLE 4.1: A description of each dataset

Dataset	Sample size	Word count range	Average word count
Trees1	340	1-560	66
Queues2	230	1-426	75
Linked1	323	1-264	55
Stacks3	222	1-419	96

The raw data contains punctuation, white space and scientific notation. Pre-processing plays a vital role in the NLP pipeline as the raw data can contain unnecessary characters and spaces which can significantly influence the outcomes of various NLP tasks. To improve the quality of data, multiple pre-processing steps were employed. Firstly, all punctuation was removed, and the text is converted to lowercase while eliminating any unnecessary spaces. Scientific notations such as O(n) and O(1) are

substituted with the English text variant "on the order of n time complexity" and "constant time complexity," respectively. Secondly, TextBlob's spell checker was used to correct for any spelling errors and lastly, lemmatization is implemented to reduce words to their base or dictionary form.

Some manipulation was carried out on the grade, including scaling it to a score out of 10 and then binning it to create a discrete value for easier evaluation of performance.

4.7.2 Vectorizing Sentences

The pre-processed data was converted into a vectorised form using sentence embedding techniques USE, BERT and Doc2Vec. The aim of this step was to represent each student response in the vector space such that similar responses would appear close to one another. "bert-base-nli-mean-tokens" is a specific variation of BERT that was used [12]. As stated earlier, sentence embeddings are chosen over word embeddings because they capture the overall meaning and context of an entire sentence, rather than focusing solely on individual words. This is important to consider given that a student's response may use the correct words but does not show understanding of the question.

4.7.3 Clustering and ECIC

The embedded responses underwent clustering using a range of methods, including k-means, hierarchical agglomerative clustering, GMM, and UMAP in combination with HDBSCAN. This step in the process was designed to group similar responses together. Although UMAP is primarily a dimensionality reduction technique, it is commonly used for text clustering in conjunction with HDBSCAN.

Hyperparameter tuning was performed on Agglomerative clustering and UMAP with HDBSCAN to optimise the methods. The Queues2 dataset was used to perform the tuning. It was then assumed that the optimal results would be the optimal results across datasets. Hyperparameter tuning was not performed on K-means and GMM due to the number of clusters remaining fixed and the other parameters

not contributing significantly to results.

To mitigate the effect of the inherent randomness in clustering techniques like K-means and Gaussian Mixture Models (GMM), a commonly adopted approach is to run the algorithm multiple times and calculate the average value of the resulting clusters. Specifically in this system, it was executed 20 times to obtain a more stable estimate of the true clustering.

Additionally, the standard deviation of the resulting clusters was recorded to provide insight into the degree of variability in the clustering outcomes. This practice helps to ensure that the final clustering solution is not unduly influenced by stochastic factors or random initialization, thereby enhancing the robustness and reliability of the results. ECIC was then applied on top of the existing clusters in order to hopefully improve on the clustering results. The ECIC algorithm can be seen in Algorithm 1.

Algorithm 1 ECIC Algorithm

Requirements: D = set of k documents, C = a set of initial cluster labels in D, M = number of clusters **Ouput**:A set of refined clusters for D
1: $iterations = 50$;
2: $TextsPerCluster = k/M$;
3: for $j = 1$ to $iterations$ do:
4: Choose a parameter P uniformly at random from the interval [P1, P2]
5: Remove outliers from each of the M clusters defined C using an outlier detection algorithm.
6: If a cluster contains more than TextsPerCluster × P texts Remove texts from that cluster uniformly at random so that exactly TextsPerCluster × P texts remain in the cluster.
7: testSet = texts removed in Steps 5 and 6, trainingSet = all the texts not in testSet.
9: Train a classifier using the trainingSet and classify the texts in testSet. This assigns a new cluster label $C(t)$ to each text t in testSet.
10: return C.

The ECIC algorithm works by taking clustered data that is already labelled and splitting the data into a training and a test set, where the training set comprises data that primarily has no outliers and a test which only contains outliers. A classifier is then trained and the test set is classified in order to improve the quality of

clusters. This process is repeated a certain number of times or till a stopping criteria is met. Given that our datasets are small, the process will always be run a predetermined number of times as the time saved is not significant.

The ECIC algorithm selects P randomly from the interval $[P_1, P_2]$, with $P_1 = 0.5$ and $P_2 = 0.95$ being the optimal values according to the original paper [28]. In this experiment, we use these values for P as well. The outlier detection algorithm and classifiers that were found to produce the optimal results in the original paper [28] are Isolation Forest and Multinomial Logistic Regression respectively and were therefore also chosen in this experiment.

Step 6 of the algorithm involves removing texts from each cluster so that each cluster contains approximately the same number of texts and is done to help reduce the bias of the classification algorithm. The algorithm is modified to use more sophisticated sentence embeddings to embed the text before it is fed to the classifier as seen in [27].

4.7.4 Sampling

Two sampling methods were investigated to select documents from each cluster to be manually marked. The first strategy involved randomly choosing documents from each cluster and then averaging the marks, applying the average mark to the entire cluster. The second strategy involved successively selecting documents based on their proximity to the mean of the cluster. The grades for the chosen documents were averaged and applied to the entire cluster. HDBSCAN automatically detected outliers.

When a submission was identified as an outlier, it implied that the submission did not share similarities with any other submissions, and the clustering method could not assign a similarity-based grade to it. Consequently, these submissions required manual marking in addition to the two sampling strategies. With both sampling strategies, the algorithm used the 'aimed_to_be_marked' value to determine how many submissions from each cluster had to be chosen. This number was incremented for testing purposes. In cases where the 'aimed_to_be_marked' value was

greater than the number of submissions in the cluster, all the submissions in the cluster were marked.

4.8 Analysis

Brown [3] evaluates human scoring of essays without the use of a rubric using Pearson correlation (PC) and Cohen's Kappa (CK). PC determines consistency by evaluating to what degree the pattern of low and high scores is similar amongst markers [26]. The equation is seen in equation 4.1. For the purpose of this research report PC refers to the correlation between the grades assigned by the automated system and the grades assigned by human raters.

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{4.1}$$

CK determines consensus by evaluating the similarity of grades between markers which can also be interpreted as how much of the similarity is not due to chance [8]. The formula is seen in equation 4.2. For the purposes of this research report CK refers to the agreement between the grades assigned by the automated system and the grades assigned by human raters.

$$K = \frac{Po - Pe}{1 - Pe} \tag{4.2}$$

where Po is the observed agreement and Pe is the expected agreement. The observed agreement, Po, is calculated as:

$$Po = \frac{\text{number of agreements}}{\text{total number of ratings}} \tag{4.3}$$

The expected agreement, Pe, is calculated as:

$$Pe = \sum_{i} \frac{n_i}{N_1} \times \frac{n_i}{N_2} \tag{4.4}$$

where n_i is the number of ratings for each category, N_1 is the total number of ratings by the first rater, and N_2 is the total number of ratings by the second rater. The linear weighting scheme assigns equal weights to all disagreements between categories. Brown [3] discovers that PC $>= 0.7$ and CK $>= 0.4$ when various markers

marked papers. Taking this into consideration the systems overall reliability will be evaluated according to PC and CK which are also deemed the primary evaluation metrics. In order to be deemed acceptable they must meet the minimum performance criteria as specified in [3]. Accuracy [15] will also be evaluated when evaluating the preprocessing on the raw data. Accuracy is defined as:

$$Accuracy = \frac{\text{Number of correctly classified instances}}{\text{Total number of instances}} \tag{4.5}$$

A more in depth analysis of the cluster quality will also be performed. The accuracy, normalized mutual information (NMI) and the Adjusted Rand Index (ARI) will be evaluated. Rand Index evaluates the similarity between cluster assignments by making pair-wise comparisons. Nevertheless, the approach fails to consider the possibility of chance, where several cluster assignments could be random. ARI addresses this by incorporating a normalization term that accounts for chance [33] The equation can be seen in equation 4.6.

$$\text{ARI} = \frac{\sum_{ij} n_{ij}2 - [\sum_i a_i 2 \sum_j b_j 2]/n2}{\frac{1}{2}[\sum_i {a_i}^2 + \sum_j {b_j}^2] - [\sum_i {a_i}^2 \sum_j {b_j}^2]/n^2} \tag{4.6}$$

where n_{ij} is the number of elements in common between cluster i and cluster j, a_i is the number of elements in cluster i, b_j is the number of elements in cluster j, and n is the total number of elements. Mutual Information (MI) evaluates the agreement between cluster assignments. NMI is MI divided by average cluster entropies [32].

$$MI(X;Y) = \sum_{y \in Y} \sum_{x \in X} p(x,y) \log \left(\frac{p(x,y)}{p(x)p(y)} \right) \tag{4.7}$$

$$\text{NMI}(X;Y) = \frac{2MI(X;Y)}{H(X) + H(Y)} \tag{4.8}$$

where $p(x,y)$ is the joint probability distribution of the variables X and Y, $p(x)$ and $p(y)$ are their marginal probability distributions, and $H(X)$ and $H(Y)$ are the entropy of X and Y, respectively which can be seen below.

$$H(X) = -\sum_{x \in X} p(x) \log p(x) \tag{4.9}$$

$$H(Y) = -\sum_{y \in y} p(y) \log p(y) \tag{4.10}$$

When evaluating a system using metrics that require a discrete value, such as Cohen Kappa and accuracy, the resulting score is often not an exact integer. In such cases, it is common practice to round up the final score to the nearest integer in order to obtain a discrete value. This will be done, in order to evaluate the performance of the system. In addition, the time taken to physically mark all the questions will be compared to the time taken whilst using the system.

Chapter 5

Results and Discussion

The Results and Discussion section of this research report presents a comprehensive analysis and interpretation of the findings obtained from the conducted research. Firstly, the results of hyper-parameter tuning on the clustering methods and pre-processing on the raw data is provided. This determines which parameters are used in the rest of the pipeline. The various sentence embeddings and clustering techniques are used in conjunction with the two sampling strategies and their results are shown. Finally, a discussion that summarises all the various combinations is given. The ECIC algorithm is then applied to the results of the pipeline and is discussed in detail.

5.1 Hyperparameter tuning

Hyperparameter tuning was performed on Agglometric clustering as well as HDBSCAN and UMAP in order to find the optimal values. The Queues2 dataset is used to perform the tuning. It is then assumed that the optimal results will be the optimal results across datasets. For HDBSCAN and UMAP, the best parameters that were found is the mininum cluster size=7, the number of components=3 and the number of neighbours=10.

However, closer inspection showed that this produced only 2 clusters which after understanding the general distribution of marks would not be helpful. The min_cluster_size was then set to 2, in order to find the optimal UMAP parameters. It was found that n_components=6 and n_neighbours=3 produced the best results. Hyperparameter tuning on Agglometric clustering showed that affinity='euclidean' and linkage='average' produced the highest Silhouette Score. However, through

manual inspection it was found that linkage='ward' produced higher PC and CK values. Therefore, the design decision is made to use 'ward'.

5.2 Preprocessing

Preprocessing is a step that is normally performed on NLP tasks in order to improve on results. A test is performed to determine the impact that punctuation removal, replacement of words, spell checking and lemmatization has on the final results produced by the model. To ensure that the test is fair, 30 clusters are chosen and 7 submissions around the center of each cluster were manually marked and the mode applied to the entire cluster.

The tables 5.1, 5.2 and 5.3 represents the results after preprocessing has been performed for the various clustering and sentence embedding combinations. It should be noted that the table represents a pipeline and each column represents another step in the pipeline. For example the column 'spelling' represents that the steps removal of punctuation and spelling having been applied to the raw data. It should also be noted that USE has no values under the column 'Data without preprocessing', given that it is sensitive to some forms of punctuation and therefore the raw data cannot be used as is.

In all cases we see an improvement when the removal of punctuation is applied. BERT has the highest improvement in terms of CK. It improves by 0,058819 for K-means clustering followed closely by Agglometric clustering which sees an improvement of 0.05843. BERT with k-means clustering see's the highest improvement in terms of PC with an increase of 0,06045 and in terms of Accuracy with a 0,054279 increase. It is seen that removal of punctuation has the most significant impact on K-means clustering coupled with BERT. However, it is noted that a comparison was unable to be performed on USE.

The next step in the pipeline that is investigated is the addition of correcting of spelling. BERT see's an improvement across all clustering techniques. In terms of CK, it has an average increase of 0.04019. In terms of PC, it has an average increase of 0.07752 and in terms of accuracy it has an average increase of 0.012537. USE see's

TABLE 5.1: CK measured after various pre-processing steps, sentence embeddings and clustering techniques are applied to text data.

Clustering technique	**Sentence embedding**	**Data without pre -processing**	**Removal of punctuation**	**Spelling**	**Lemmat-ization**
GMM	USE	none	0.466679	0.470693	0.460512
	BERT	0.374342	0.391599	0.395488	0.395942
	Doc2Vec	0.359947	0.367914	0.322947	0.313343
K-means	USE	none	0.475719	0.473608	0.477346
	BERT	0.318671	0.37749	0.402072	0.395699
	Doc2Vec	0.353484	0.38336	0.320094	0.305021
Agglometric	USE	none	0.521693	0.52985	0.54845
	BERT	0.329504	0.387934	0.480038	0.381935
	Doc2Vec	0.327862	0.334466	0.307343	0.276765
HDBSCAN	USE	none	0.457988	0.476148	0.484094
	BERT	0.544771	0.574641	0.487314	0.478839
	Doc2Vec	0.238813	0.296424	0.27882	0.380479

TABLE 5.2: PC measured after various pre-processing steps, sentence embeddings and clustering techniques are applied to text data.

Clustering technique	**Sentence embedding**	**Data without pre -processing**	**Removal of punctuation**	**Spelling**	**Lemmat-ization**
GMM	USE	none	0.619818	0.627349	0.610794
	BERT	0.480768	0.483665	0.490628	0.492579
	Doc2Vec	0.433853	0.448691	0.408998	0.368168
K-means	USE	none	0.625849	0.640548	0.633745
	BERT	0.400331	0.460781	0.509715	0.495569
	Doc2Vec	0.428397	0.46501	0.417916	0.356475
Agglometric	USE	none	0.666617	0.67161	0.688047
	BERT	0.415943	0.431035	0.607703	0.433305
	Doc2Vec	0.430037	0.431761	0.4151	0.284141
HDBSCAN	USE	none	0.672949	0.736936	0.754963
	BERT	0.797219	0.824956	0.734977	0.731413
	Doc2Vec	0.468686	0.505761	0.489815	0.583498

an improvement when coupled with GMM and Agglometric clustering but sees a decline when coupled with k-means. The average change in CK is 0.000971 and

TABLE 5.3: Accuracy measured after various pre-processing steps, sentence embeddings and clustering techniques are applied to text data.

Clustering technique	**Sentence embedding**	**Data without pre -processing**	**Removal of punctuation**	**Spelling**	**Lemmat-ization**
GMM	USE	none	0.421847	0.428829	0.416667
	BERT	0.401351	0.417793	0.436937	0.429392
	Doc2Vec	0.390766	0.41509	0.365991	0.364302
K-means	USE	none	0.430405	0.427477	0.42455
	BERT	0.360135	0.414414	0.428378	0.431081
	Doc2Vec	0.381757	0.417568	0.363514	0.362162
Agglometric	USE	none	0.454955	0.468468	0.463964
	BERT	0.355856	0.445946	0.45045	0.454955
	Doc2Vec	0.355856	0.364865	0.36036	0.369369
HDBSCAN	USE	none	0.313043	0.291304	0.252174
	BERT	0.321739	0.33913	0.291304	0.269565
	Doc2Vec	0.204348	0.217391	0.173913	0.291304

the average change in PC is 0.000971. Though there is a decrease with k-means the change is so minimal that the design decision is chosen to keep the spell checking for the cases where there is an improvement. Doc2Vec see's a decrease across all clustering methods. CK decreases by 0.04511 and PC decreases by 0.03448.

The last step in the pipeline that is investigated is lemmatization. The results were sporadic with there sometimes being an increase and in other cases there being an decrease for all three metrics for the various clustering and embedding techniques. Overall a decline is seen with CK value decreasing by 0.01635 and PC decreasing by 0.04742, This could be due to sentence embeddings being used instead of word embeddings and by lemmatizing the words, the overall meaning of the sentence can change.

The same steps in the pipeline were applied to HDBSCAN. It differs in comparison to the other clustering methods in that the number of clusters is not predefined but is automatically determined by the algorithm based on what the cluster_min_size is set to and the format of the data. In the tests cluster_min_size is set to 2 at all

stages, however the number of clusters produced at each step varies which can skew the PC and CK value. For example, the checking of spelling is seen to decrease the PC and CK value for BERT and Doc2Vec but increase performance for USE. However, closer inspection reveals that fewer clusters were generated using BERT which could imply that the decrease is as a consequence of that and not the removal of punctuation. When the step was run on a greater range of values, the decrease was not always seen. The effect of the other steps in the pipeline followed the same trend as the other clustering techniques whereby spelling improved results, except in the case of Doc2Vec, and lemmatization decreased results. It is therefore assumed that the removal of punctuation may indeed improve results, however nothing conclusive can be said about this.

After evaluating the results the design decision is taken to apply the removal of punctuation to all combinations, to apply a spell checker to USE and BERT but to not apply it to Doc2Vec with all the different clustering techniques. It is also decided to completely remove lemmatization given that the overall effect is negative. Preprocessing can have significant impact on the results achieved. For example K-means and BERT saw an increase in Cohen's Kappa of 0.1 when applying the steps removal of punctuation and spell checking. It is therefore a crucial step in building an automated marking system.

5.3 Clustering with Random Sampling

An investigation was performed to cluster the data using various clustering and sentence embedding techniques and then randomly choose submissions from each cluster to mark. A summary of the results can be seen in Table 5.4. It can be seen that this sampling technique did not reach the minimum PC and CK value required using the Queues2 dataset and this trend is seen across the other datasets.

TABLE 5.4: Summary of results using random sampling in conjunction with various clustering and sentence embedding combinations on the Queues2 dataset

Clustering technique	Embedding	PC range	CK range
GMM	BERT	-0.05159 to 0.533798	-0.00021-0.319810
	USE	-0.00815 to 0.495006	-0.00547 to 0.305368
	Doc2vec	0.228215 to 0.575537	0,006037 to 0.36638
k-means	BERT	-0,0682 to 0.609571	-0.0018 to 0.355341
	USE	-0,10634 to 0.460931	-0.00783 to 0.25118
	Doc2vec	0.20077 to 0.594097	0.003389 to 0.369677
Agglomerative	BERT	-0.32034 to 0.635632	-0.00445 to 0.380674
	USE	0.052319 to 0.579985	-0.00735 to 0.358347
	Doc2vec	0.238621 to 0.615966	0.006248 to 0.387561
HDBSCAN	BERT	0.184334 to 0.628345	0.062763 to 0.432335
	USE	0.181655 to 0.468609	0.08644 to 0.345995
	Doc2vec	0.06695 to 0.278046	0.040816 to 0.186729

5.4 Clustering with Center Sampling

Another investigation was performed whereby the data was grouped together using various clustering and sentence embedding and then submissions are successively chosen depending on their distance to the center of the cluster. The results for each clustering method is broken down and final comparison between all methods is made. It should be noted that the number of documents that were aimed to be marked in each cluster will be referred to as K in the below tables.

5.4.1 GMM results

A table representing where combinations achieve the necessary performance metrics can be seen in Table A.1. In the Stacks3 and Linked1 data set, the only sentence embedding that yields an acceptable result is USE and BERT respectively. In the Queues2 and Trees1 data sets, acceptable results are achieved for both USE and BERT. It should be noted that no combination with Doc2Vec achieved acceptable results.

TABLE 5.5: Examples where the minimum percentage of submissions are marked that achieve acceptable PC and CK results without considering standard deviation using center sampling in conjunction with GMM and various sentence embeddings.

Dataset	Embedding	Components	K	% marked	PC	CK	PC std	CK std
Stacks3	USE	40	5	55	0.7008	0.4409	0.0273	0.0323
Linked1	BERT	40	7	67	0.7027	0.4210	0.0262	0.0229
Queues2	BERT	40	2	32	0.7133	0.4516	0.0252	0.0259
	USE	40	2	31	0.7028	0.4541	0.0279	0.0245
Trees1	BERT	40	5	76	0.7201	0.4205	0.0237	0.0227
	USE	40	3	46	0.7131	0.4345	0.0323	0.0217

Table 5.5 summarises the examples where the minimum percent of submissions is marked that achieve acceptable results for each sentence embedding. What is noticeable for data sets Queues2 and Trees1 is that USE results in a lower percentage being marked in comparison to BERT in order to achieve acceptable results. For Queues2, BERT requires 32% marked whilst USE only requires 31% marked. In Trees1, BERT requires 76% whilst USE only requires 46%. The minimum percentage required across data sets ranges from 31% to 75%. For USE it ranges from 31% to 55% and for BERT it ranges from 32% to 75%.

In all cases 40 GMM components are needed to reach adequate performance whilst maintaining the minimum percentage marked but the number of submissions aimed to be marked varies. However, due to the randomness present in GMM's, the standard deviation of results needs to be considered. The standard deviation is subtracted from the mean PC value and the CK value. Stacks3 and Linked1 do not meet the minimum criteria, whilst Queues2 and Trees1 do. The percent required to be marked varies from 45% to 75%.

Table 5.6 highlights the examples where the maximum PC and CK values are achieved. The maximum PC value ranges from 0.708109 to 0.755311 and the maximum CK values range from 0.416972 to 0.493423. The average PC and CK value across all data sets can be seen in Table 5.7. It should be noted that this table is the average of all results and not just the combinations that met the performance metrics. For PC, BERT produces a higher value in three quarters of the cases and for CK there is no

TABLE 5.6: Examples where the maximum PC and CK values are achieved using center sampling in conjunction with GMM and various sentence embeddings.

Dataset	Embedding	Components	K	% marked	PC	CK
Stacks3	USE	40	9	76	0,7170	0,4519
Linked1	BERT	40	9	77	0,7081	0,4203
Queues2	BERT	40	9	88	0,7553	0,4934
	USE	40	9	82	0,7524	0,4687
Trees1	BERT	40	9	95	0,7348	0,4170
	USE	40	9	85	0,7429	0,4386

TABLE 5.7: Average PC and CK score using center sampling in conjunction with GMM and various sentence embeddings.

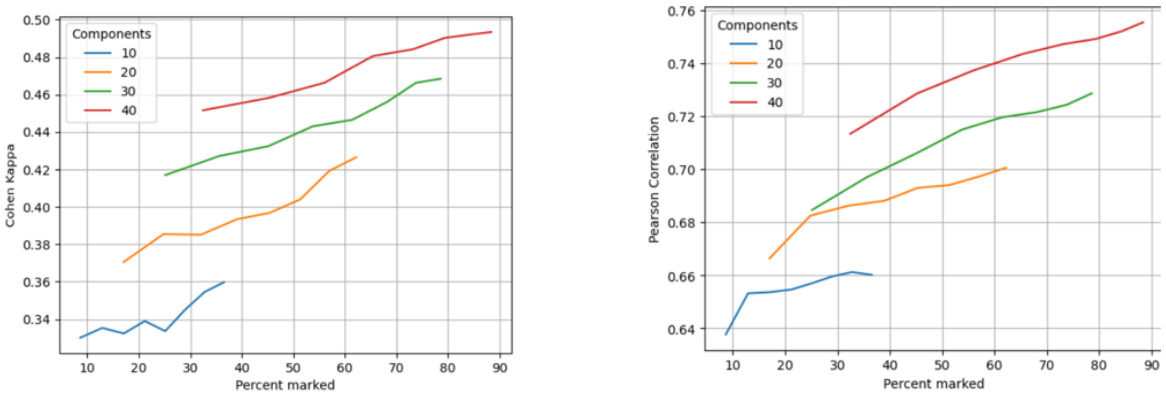
Dataset	Embedding	Average PC	Average CK
Trees1	BERT	0.669171	0.361912
	USE	0.661864	0.373649
Queues2	BERT	0.698945	0.415152
	USE	0.659292	0.380679
Linked1	BERT	0.59601	0.343452
	USE	0.471421	0.252296
Stacks3	BERT	0,507374	0,281688
	USE	0.650739	0.375814

clear winner.

The experiment also showed that increasing the number of components has more of an impact on both the PC score as well as the CK score, when a similar percentage of submissions is marked. This can be seen in Figure 5.1 when approximately 40% of submissions are marked, the PC score for 40 components is 0.728518 whilst for 30 components it is 0.705968 and for 20 components it is 0.692918. The CK score for 40 components is 0.458229 whilst for 30 components it is 0.432399 and for 20 components it is 0.396775.

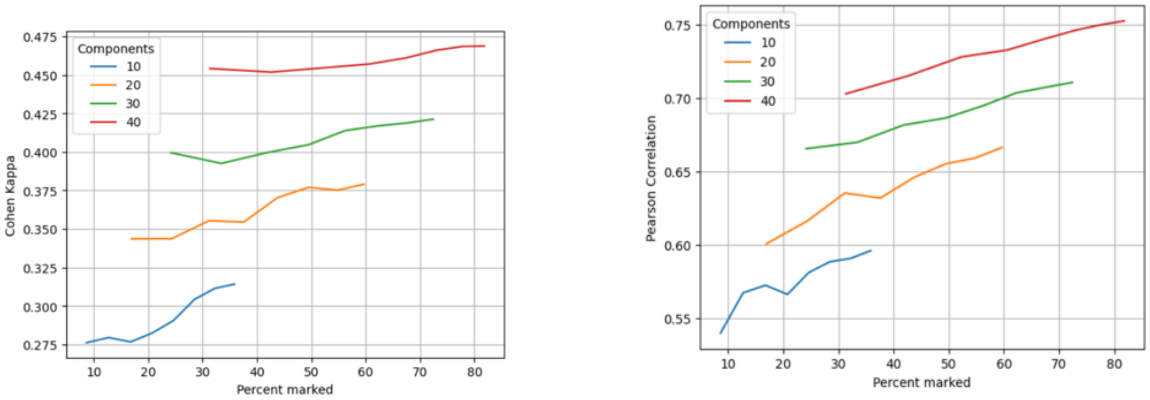
The same can also be seen using USE, as seen in Figure 5.2 where approximately 50% of submissions are marked. The PC score for 40 components is 0.739019 whilst for 30 components it is 0.705587 and for 20 components it is 0.684874. The CK score

for 40 components is 0.457414 whilst for 30 components it is 0.427964 and for 20 components it is 0.389416. This trend is seen through all four data sets.



(A) CK (B) PC

FIGURE 5.1: Plot comparing the number of responses marked with PC and CK scores using the combination of GMM and BERT on the Queues2 data set



(A) CK (B) PC

FIGURE 5.2: Plot comparing the number of responses marked with PC and CK scores using the combination of GMM and USE on the Queues2 dataset

5.4.2 K-means

A table representing where combinations achieve the necessary performance metrics can be seen in Table B.1. The table reveals that, in the Stacks3 and Linked1 data

sets, only the USE and BERT sentence embeddings, respectively, produced acceptable results. However, for the Queues2 and Trees1 data sets, acceptable results were obtained using both USE and BERT embeddings. It is worth noting that none of the combinations with Doc2Vec yielded satisfactory results.

TABLE 5.8: Examples where the minimum percent of submissions are marked that achieve acceptable results without considering standard deviation using center sampling in conjunction with Kmeans and various sentence embeddings.

Dataset	**Embedding**	**Components**	**K**	**% marked**	**PC**	**CK**	**PC std**	**CK std**
Stacks3	USE	40	5	54	0.7033	0.4401	0.0293	0.0263
Linked1	BERT	40	6	60	0.7023	0.4289	0.0224	0.0288
Queues2	BERT	40	2	33	0.7113	0.4488	0.0209	0.0201
	USE	40	2	31	0.7096	0.4465	0.0290	0.0344
Trees1	BERT	40	3	54	0.7006	0.4226	0.0363	0.0228
	USE	40	3	44	0.7188	0.4414	0.0298	0.0268

Table 5.8 provides a summary of the instances where the minimum percentage of marked submissions is reached to obtain satisfactory results for each sentence embedding. What is noticeable for data sets Queues2 and Trees1 is that USE results in a lower percentage being marked in comparison to BERT in order to achieve acceptable results. For Queues2, BERT requires 33% marked whilst USE only requires 31% marked. In Trees1, BERT requires 54% whilst USE only requires 44%. The minimum percentage required across data sets ranges from 31% to 60%. For USE it ranges from 31% to 54% and for BERT it ranges from 33% to 60%.

To achieve satisfactory performance while maintaining the minimum percentage of marked documents, 40 components are required in all cases, although the targeted number of marked documents differs. However, it's important to consider the standard deviation of results due to the inherent randomness of the k-means algorithm. The mean PC value and CK value are adjusted by subtracting the standard deviation. While Queues2 and Trees1 datasets meet the minimum requirements, Stacks3 and Linked1 datasets do not. The percentage of marked submissions needed ranges from 45% to 83%.

TABLE 5.9: Examples where the maximum PC and CK values are achieved using center sampling in conjunction with Kmeans and various sentence embeddings.

Dataset	Embedding	Components	K	% marked	PC	CK
Stacks3	USE	40	9	75	0.7177	0.4545
Linked1	BERT	40	9	78	0.7129	0.4224
Queues2	BERT	40	9	88	0.7588	0.4909
	USE	40	9	81	0.7620	0.4785
Trees1	BERT	40	9	94	0.7428	0.4167
	USE	40	9	83	0.7457	0.4392

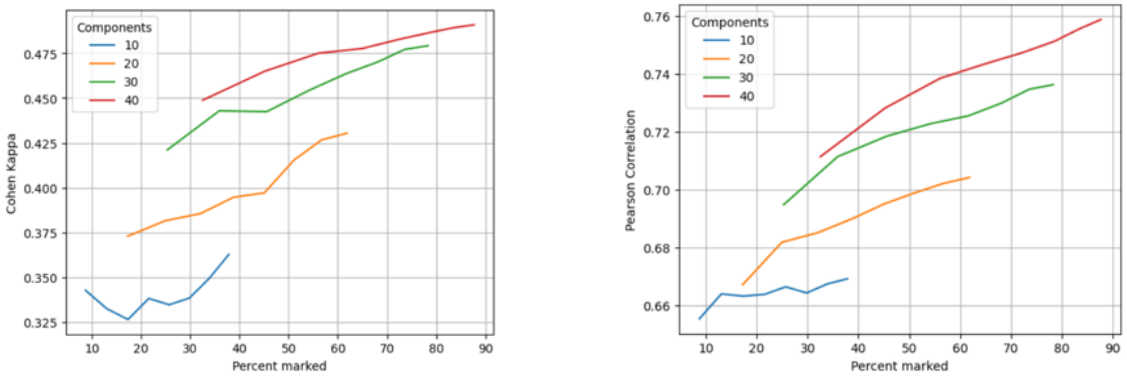
Table 5.9 outlines the instances where the highest PC and CK values are attained. The maximum PC value ranges from 0.7129 to 0.7620, while the maximum CK value ranges from 0.4167 to 0.4909. Meanwhile, Table 5.10 presents the average PC and CK value across all datasets. It is important to note that this table shows the average of all results and not just the combinations that met the performance metrics. Regarding PC and CK values, BERT has a higher value in two out of four cases, while USE has a higher value in the other two. Therefore, no clear winner can be determined.

TABLE 5.10: Average PC and CK score using center sampling in conjunction with Kmeans and various sentence embeddings.

Dataset	Embedding	Average PC	Average CK
Tree1	BERT	0,665452	0,359393
	USE	0,670473	0,378198
Queues2	BERT	0,704515	0,418657
	USE	0,677480	0,394791
Linked1	BERT	0,60203	0,347048
	USE	0,457008	0,244893
Stacks3	BERT	0,522293	0,288451
	USE	0,662802	0,385416

The experiment also showed that increasing the number of components has more of an impact on both the PC score as well as the CK score, when a similar percentage of submissions are marked. This can be seen in Figure 5.3 when approximately 40% of submissions are marked, the PC score for 40 components is 0.728518 whilst

for 30 components it is 0.705968 and for 20 components it is 0.692918. The CK score for 40 components is 0.465093 whilst for 30 components it is 0.442424 and for 20 components it is 0.397153. The same can also be seen using USE, as seen in Figure 5.4 where approximately 50% of documents is marked. The PC score for 40 components is 0.739019 whilst for 30 components it is 0.705587 and for 20 components it is 0.682596. The CK score for 40 components is 0.457414 whilst for 30 components it is 0,427964 and for 20 components it is 0.391699. This trend is seen through all four data sets.



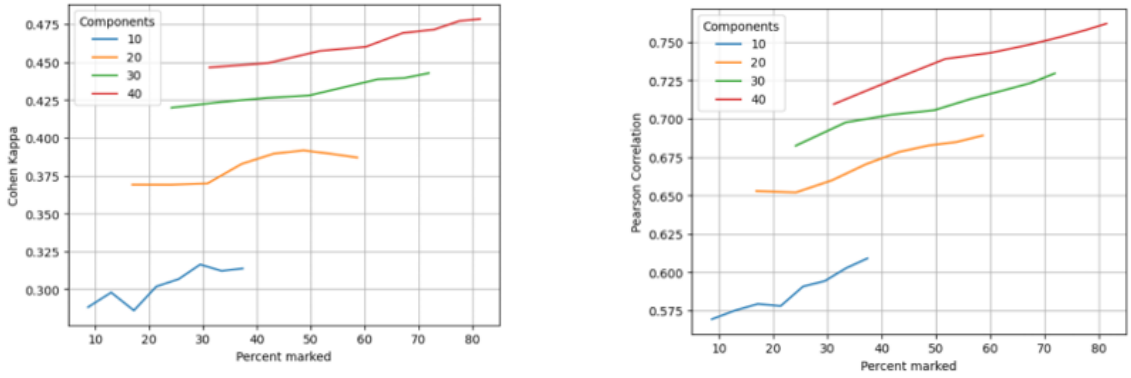
(A) CK (B) PC

FIGURE 5.3: Plot comparing the number of responses marked with PC and CK scores using the combination of kmeans and BERT on the Queues2 data set

5.4.3 Hierarchical Agglomerative clustering

A table representing where combinations achieve the necessary performance metrics can be seen in Table C.1. In the Stacks3 and Linked1 data set, the only sentence embedding that yields an acceptable result is USE and BERT respectively. In the Queues2 and Trees1 data sets, acceptable results are achieved for both USE and BERT. It should be noted that no combination with Doc2Vec achieved acceptable results.

Table 5.11 summarises the examples where the minimum percent of submissions are marked that achieve acceptable results for each sentence embedding. In this instance, USE does not always have a lower percentage in comparison to BERT. In



(A) CK (B) PC

FIGURE 5.4: Plot comparing the number of responses marked with PC and CK scores using the combination of kmeans and USE on Queues2 data set

TABLE 5.11: Examples where the minimum percent of submissions is marked that achieve acceptable results using center sampling in conjunction with Hierarchical Agglometric clustering and various sentence embeddings.

Dataset	**Embedding**	**Components**	**K**	**% marked**	**PC**	**CK**
Stacks3	USE	30	3	34	0,7016	0,4378
Linked1	BERT	30	9	68	0,7025	0,4194
Queues2	BERT	30	2	26	0,7150	0,4759
	USE	30	3	36	0,7034	0,4034
Trees1	BERT	40	3	57	0,7197	0,4111
	USE	40	2	37	0,7372	0,4533

Queues2, BERT requires 26% of the question papers to be marked whilst USE requires 36%. The minimum percentage required across data sets ranges from 26% to 68%. For USE it ranges from 34% to 37% and for BERT it ranges from 26% to 68%. The components required varies between 30 and 40 components.

Table 5.12 highlights the examples where the maximum PC and CK values are achieved. The maximum PC value ranges from 0.7143 to 0.7606 and the maximum CK values range from 0.4157 to 0.4983. The average PC and CK value across all data sets can be seen in Table 5.13. It should be noted that this table is the average of all results and not just the combinations that met the performance metrics. For

TABLE 5.12: Examples where the maximum PC and CK values are achieved using center sampling in conjunction with Hierarchical Agglometric clustering and various sentence embeddings.

Dataset	Embedding	Components	K	% marked	PC	CK
Stacks3	USE	40	9	73	0,7505	0,4983
Linked1	BERT	40	9	78	0,7143	0,4300
Queues2	BERT	40	2	33	0,7354	0,4933
	USE	40	9	87	0,7606	0,4666
Trees1	BERT	40	9	98	0,7369	0,4157
	USE	40	9	87	0,7727	0,4688

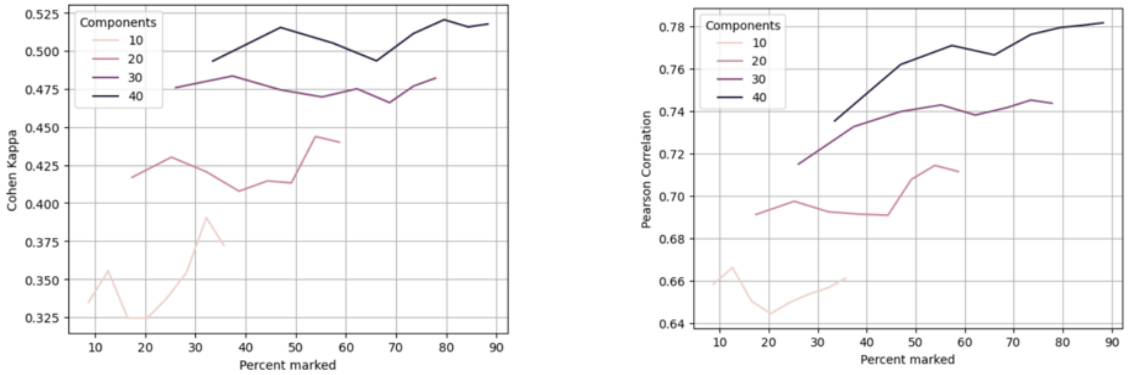
PC and CK, BERT has a higher value in half the cases and USE has a higher value in half cases.

TABLE 5.13: Average PC and CK score using center sampling in conjunction with Hierarchical Agglometric clustering and various sentence embeddings.

Dataset	Embedding	Average PC	Average CK
Tree1	BERT	0.649226	0.361687
	USE	0.693325	0.409127
Queues2	BERT	0.715241	0.439280
	USE	0.680406	0.380241
Linked1	BERT	0.604156	0.348659
	USE	0.534696	0.290923
Stacks3	BERT	0.548297	0.332380
	USE	0.692026	0.419029

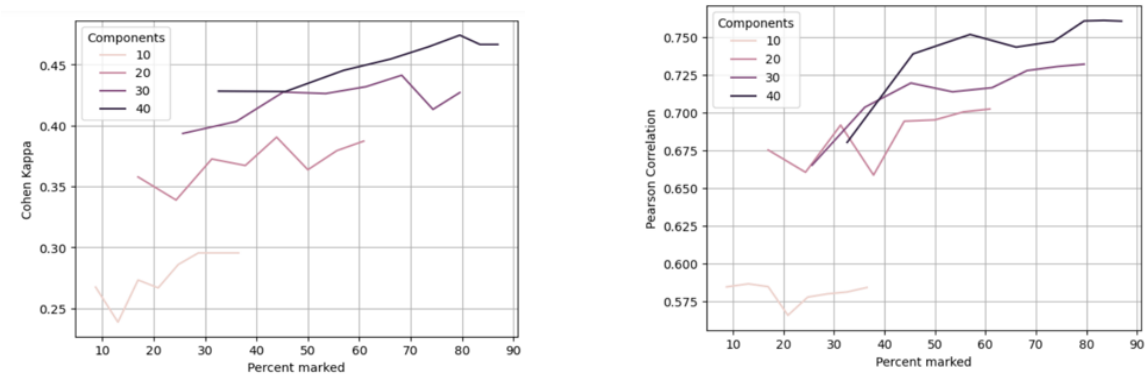
The experiment also showed that increasing the number of components has more of an impact on both the PC score as well as the CK score, when a similar percentage of submissions are marked. This can be seen in Figure 5.5 when approximately 50% of submissions are marked, the PC score for 40 components is 0.728518 whilst for 30 components it is 0.705968 and for 20 components it is 0.692918. The CK score for 40 components is 0.465093 whilst for 30 components it is 0.442424 and for 20 components it is 0.397153. The same can also be seen using USE, as seen in Figure 5.6 where approximately 50% of submissions are marked. The PC score for 40 components is 0.770968 whilst for 30 components it is 0.742901 and for 20

components it is 0,707721. The CK score for 40 components is 0.505113 whilst for 30 components it is 0.469701 and for 20 components it is 0.413251. This trend is seen through all four data sets.



(A) CK (B) PC

FIGURE 5.5: Plot comparing the number of responses marked with PC and CK scores using the combination of agglomerative clustering and BERT on Queues2 data set



(A) CK (B) PC

FIGURE 5.6: Plot comparing the number of responses marked with PC and CK scores using the combination of agglomerative clustering and USE on the Queues2 data set

5.4.4 HDBSCAN

HDBSCAN differs to the other clustering methods that were investigated earlier in that the number of clusters is not defined. The only variable that is manually varied

is the minimum cluster size which subsequently results in HDBSCAN automatically identifying the number of components. Table D.1 highlights the combination of minimum clustering size and various sentence embedding which resulted in the necessary performance metrics being met.

In the Stacks3 and Trees1 dataset the only embedding method the yields adequate results is USE and BERT respectively. In the Linked1 and Queues2 dataset, both USE and BERT yield adequate results. It should be noted that Doc2Vec produces significantly lower results and never achieves adequate results. Varying the minimum cluster size from 2 to 5 yields components of between 11 to 54.

TABLE 5.14: Examples where the minimum percent of submissions are marked that achieve acceptable results using center sampling in conjunction with UMAP and HDBSCAN combined with various sentence embeddings.

Dataset	**Embedding**	**Min-Cluster Size**	**Components**	**K**	**% marked**	**PC**	**CK**
Stacks3	USE	3.0	22	3	30	0.7409	0.4932
Linked1	BERT	4.0	24	3	22	0.7396	0.5010
	USE	5.0	20	9	52	0.7016	0.4751
Queues2	BERT	5.0	14	4	24	0.7050	0.4935
	USE	5.0	11	7	33	0.7059	0.4600
Trees1	BERT	2.0	34	2	37	0.7408	0.4800

Table 5.14 displays the instances where the minimum percentage of marked submissions was required to achieve satisfactory results for each sentence embedding. The minimum percentage of marked submissions needed across datasets ranges from 22% to 52%, with an average requirement of 33% of the questions being marked. For the USE embedding, the minimum percentage required varied from 30% to 52%, whereas for the BERT embedding, it varied from 22% to 37%. In cases where both USE and BERT yielded acceptable results for a single dataset, BERT required a lesser number of marked submissions. For instance, in the Queues2 dataset, USE needed 33% of the questions marked, whereas BERT only required 24%. However, in cases where only one sentence embedding produced acceptable results, it varied between USE and BERT. Out of the six cases, the minimum cluster size required

was 5 in three instances. It's worth noting that setting the minimum number of clusters to 5 didn't always yield adequate results, as demonstrated in the Linked1 dataset. Nevertheless, across all datasets that achieved satisfactory performance, the minimum cluster size was consistently set to 2. When set to 2, the minimum number of submissions required varied from 34% to 74% for USE and from 33% to 58% for BERT.

TABLE 5.15: Examples where the maximum PC and CK values are achieved using center sampling in conjunction with UMAP and HDBSCAN combined with various sentence embeddings

Dataset	**Embedding**	**Min-Cluster Size**	**Components**	**K**	**% marked**	**PC**	**CK**
Stacks3	USE	3.0	38	8	83	0.7809	0.5394
Linked1	BERT	4.0	54	8	83	0.7881	0.5079
	USE	5.0	59	9	89	0.7184	0.4608
Queues2	BERT	5.0	37	9	84	0.7423	0.4788
	USE	5.0	33	8	75	0.7548	0.4844
Trees1	BERT	5.0	34	9	89	0.8118	0.4953

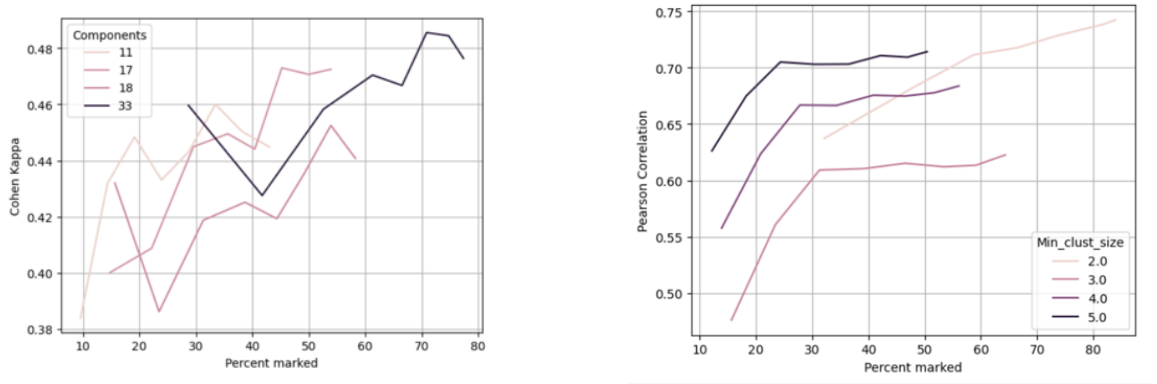
Table 5.15 highlights the examples where the maximum PC and CK values are achieved. The maximum PC value ranges from 0.718389 to 0.811845 and the maximum CK values range from 0.460761 to 0.539388. The average PC and CK value across all data sets can be seen in Table 5.16. It should be noted that this table is the average of all results and not just the combinations that met the performance metrics. For PC and CK, BERT has a higher value in 2/4 cases and USE has a higher value in 2/4 cases.

Increasing the minimum distance between clusters leads to the creation of fewer clusters. However, having a higher number of clusters doesn't necessarily result in better PC and CK scores. This can be observed in the figure 5.7, where approximately 40% of submissions are marked. It can be seen that 37 components (with a minimum cluster size of 2) gives a PC of 0.682005 and a CK score of 0.434214. On the other hand, having 18 components (with a minimum cluster size of 3) results in a PC of 0.610379 and a CK score of 0.29099, while having 16 components (with a minimum cluster size of 4) results in a PC of 0.675566 and a CK score of 0,460001.

TABLE 5.16: Average PC and CK score using center sampling in conjunction with UMAP and HDBSCAN combined with various sentence embeddings

Dataset	Embedding	Average PC	Average CK
Tree1	BERT	0.693469	0.356536
	USE	0.432175	0.261665
Queues2	BERT	0.662024	0.440022
	USE	0.696684	0.443683
Linked1	BERT	0.736165	0.473615
	USE	0.659967	0.441716
Stacks3	BERT	0.506389	0.336387
	USE	0.692835	0.463132

Having 14 components (with a minimum cluster size of 5) leads to a PC score of 0.710716 and a CK score of 0.514819.The same trend can be seen when USE is used as can be seen in figure 5.8



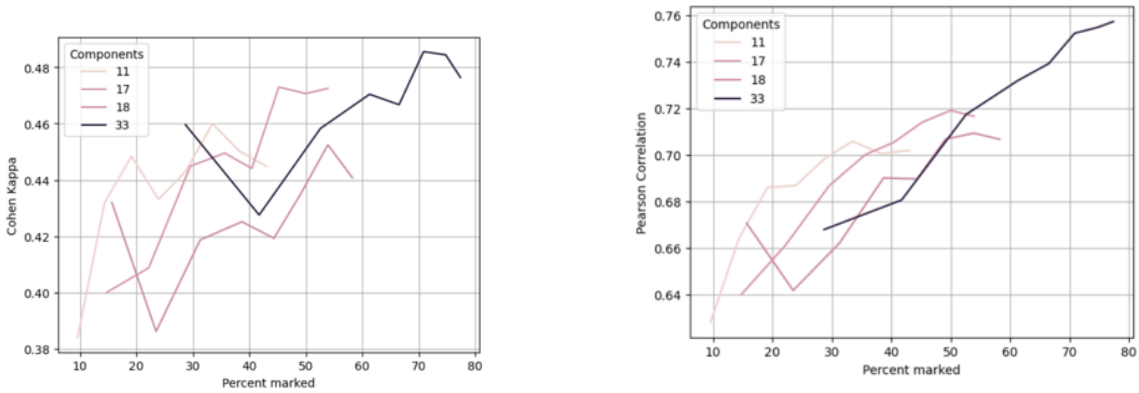
(A) CK

(B) PC

FIGURE 5.7: Plot comparing the number of responses marked with PC and CK scores using the combination of HDBSCAN and BERT on the Queues2 data set

5.5 Overall comparison

An automated marking system can be evaluated on how much time it would save a marker and on how reliable the system is. As noted earlier a system is deemed to be acceptable if it obtains a PC score $>= 0.7$ and a CK score $>= 0.4$. Various



(A) CK (B) PC

FIGURE 5.8: Plot comparing the number of responses marked with PC and CK scores using the combination of HDBSCAN and USE on the Queues2 data set

clustering and sampling methods were investigated. It was found that randomly selecting submissions from each cluster did not achieve acceptable results however successively choosing documents around the center of each cluster did.

Table 5.17 summarises how each combination of clustering algorithm and sentence embedding performed on the various data sets. Green represents that it was able to meet the minimum performance criteria and red represents that it did not. GMM, K-means and Agglomerative clustering follow the same trend across datasets, however HDBSCAN varies. Trees1 and Queues2 datasets were the most text heavy questions, having equations and other computational jargon to a minimum.

For both these datasets BERT and USE produced acceptable results across datasets except in the case of HDBSCAN. Linked1 and Stacks3 have significantly more mathematical notation and other computational jargon and in these cases the system did not work as effectively. From this we can conclude that the system is not as effective on questions that are mathematically and notation heavy. What is also noted is that the sentence embedding that is most successful in these cases tends to vary depending on the dataset which can be seen in Linked1, where BERT was the most successful and for Stacks3, USE was the most successful across clustering techniques.

TABLE 5.17: Summary of which sentence embedding and clustering technique achieved acceptable performance for both PC and CK across datasets.

Dataset	Embedding	GMM	K-means	Agglomerative	HDBSCAN
Tree1	BERT				
	USE				
	Doc2Vec				
Queues2	BERT				
	USE				
	Doc2Vec				
Linked1	BERT				
	USE				
	Doc2Vec				
Stacks3	BERT				
	USE				
	Doc2Vec				

The first aspect of the experiment that was investigated was what percentage of submissions are required to achieve acceptable results. Table 5.18 summarises this for various sentence embeddings and clustering methods and Table 5.19 further breaks this done for each dataset. HDBSCAN is found to have the smallest range with an average of 33%, followed by K-means then Agglomerative clustering and finally GMM which is the worst out of all clustering methods. All methods achieve an average minimum value that is below 51%.

TABLE 5.18: Summary of the minimum percentage of submissions to be marked to reach PC and CK performance requirements.

	GMM	K-means	Agglometric	HDBSCAN	HDBSCAN (min_clust=2)
Overall range(%)	31-76	31-60	26-68	24-52	34-74
Average(%)	51	41	43	33	48
USE range(%)	31-55	31-54	34-37	24-52	34-74
BERT range(%)	32-75	33-60	26-68	22-37	33-58

We further breakdown the minimum percentage of submissions required to be

marked according to sentence embedding. For GMM, k-means and Agglomerative clustering, BERT has a wider range in comparison to USE. But for HDBSCAN, USE has a wider range. Looking at the lower bound of the range, USE has a lower value for both GMM and Kmeans, however BERT has a lower value for Agglomerative and HDBSCAN. Looking at the upper bound of the range, USE has a lower value for GMM, K-means and Agglomerative whilst BERT has a lower value for HDBSCAN.

In datasets where both USE and BERT were deemed to produce acceptable results, it was seen that USE required less submissions than BERT to be marked for both GMM and K-means. In Agglomerative clustering there was no distinctly better performing sentence embedding. However for HDBSCAN, BERT required less submissions than USE.

A balance must be found between the minimum percent required (lower bound of the range) as well as the range itself. Taking this into consideration, HDBSCAN with BERT produces the best results followed by Agglomerative with USE. Both at a maximum only require 37% of submissions to be marked.

For GMM, K-means and Agglomerative clustering, 40 clusters were required to achieve acceptable results across datasets, however there was no consistent 'Aimed to be marked' value that appeared to be effective across datasets. As shown earlier for HDBSCAN, the minimum cluster size being set to 5 produced the optimal minimum results in 50% of the cases across datasets. However, the minimum cluster size being set to 2 was the only value consistently seen to produce acceptable results even if it wasn't the best performing combination.

Setting the minimum cluster size to 2 results in an average of 48% of questions being required to be marked with USE ranging from 34-74% and BERT ranging from 33-58%. Again there was no 'Aimed to be mark' value that appeared to be effective across datasets for HDBSCAN.

This needs to be factored in when deciding which combination produces the best minimum results given that the system needs to be generalised to work effectively

for various questions. Whilst hyperparameter tuning could be performed on each dataset to retrieve the optimal min_cluster size given that the datasets are small and therefore may not take too long, it is not ideal to be doing this in a real world system. Taking this into consideration Agglomerative with USE performs the best.

TABLE 5.19: The minimum percentage of submissions to be marked to reach PC and CK performance requirements per dataset

Clustering	**Dataset**	**Embedding**	**Components**	**K**	**% marked**	**PC**	**CK**
GMM	Stacks3	USE	40	5	55	0.7008	0.4409
	Linked1	BERT	40	7	67	0.7027	0.4210
	Queues2	BERT	40	2	32	0.7133	0.4516
		USE	40	2	31	0.7028	0.4541
	Trees1	BERT	40	5	76	0.7201	0.4205
		USE	40	3	46	0.7131	0.4345
K-means	Stacks3	USE	40	5	54	0.7033	0.4400
	Linked1	BERT	40	6	60	0.7023	0.4289
	Queues2	BERT	40	2	33	0.7113	0.4488
		USE	40	2	31	0.7095	0.4465
	Trees1	BERT	40	3	54	0.7006	0.4226
		USE	40	3	44	0.7188	0.4413
Agglomerative	Stacks3	USE	30	3	34	0.7016	0.4378
	Linked1	BERT	30	9	68	0.7025	0.4193
	Queues2	BERT	30	2	26	0.7150	0.4759
		USE	30	3	36	0.7035	0.4033
	Trees1	BERT	40	3	57	0.7197	0.4110
		USE	40	2	37	0.7372	0.4533
HDBSCAN	Stacks3	USE	22	3	30	0.7409	0.4932
	Linked1	BERT	24	3	22	0.7396	0.5010
		USE	20	9	52	0.7017	0.4750
	Queues2	BERT	14	4	24	0.7050	0.4935
		USE	11	7	33	0.7059	0.4599
	Trees1	BERT	34	2	37	0.7409	0.4800

The second aspect of the experiment that was investigated was the maximum PC and CK value that was reached for each dataset. Table 5.20 summarises this. HDBSCAN achieved the highest performance followed by Agglomerative clustering then K-means and finally GMM.

TABLE 5.20: Summary of the maximum PC and CK achieved for various clustering techniques

Metric	GMM	K-means	Agglometric	HDBSCAN
PC range	0.7081-0.7553	0.7129-0.7620	0.7143-0.7727	0.7184-0.8118
CK range	0.4169-0.4934	0.4157-0.4909	0.4157-0.4983	0.4608-0.5394

The third aspect of the experiment that was investigated was evaluating the average value obtained across datasets. Table 5.21 summarises the range of average scores achieved across datasets. HDBSCAN has the largest range whilst Agglomerative has the smallest range for both PC and CK. For GMM, BERT produced better results than USE, however for the remaining clustering algorithms there wasn't a sentence embedding that was distinctly better.

TABLE 5.21: Summary of the average PC and CK achieved for various clustering techniques.

Metric	GMM	K-means	Agglometric	HDBSCAN
PC range	0.4714-0.6989	0.4570-0.7045	0.5347-0.7152	0.4322-0.7362
CK range	0.2523-0.4151	0.2449-0.4187	0.2909-0.4393	0.2616-0.4736

Testing is performed using other popular clustering evaluation metrics in order to try and gain more insight into the clusters themselves. The results, as seen in 5.22, show that there are minimal differences between clustering techniques. Generally, Ajusted Rand Index and NMI values of above 0.5 are considered to be adequate however the values fall below this and accuracy only reaches 0.3. This indicates that the clusters themselves are not of high quality.

Clustering short text is a difficult task given that the vectors produced in the vector space tend to be sparse given the limited information available. On top of this, the datasets are small with there only being a few hundred responses for each question. To add to this difficult problem, in most topic modelling and short text clustering problems there are distinct topics which would be spread out in the vector space.

For example, in social media data, there may be topics like technology and animals which could be far away from one another in the vector space making it much easier to cluster. However, in this problem the semantic differences in responses is a lot more subtle and difficult to pick up which makes clustering them more difficult. There are no baseline values for this kind of problem and therefore it is difficult to evaluate the system on these metrics and make any conclusive remarks. However, the system reaches the minimum PC and CK value which imply the system may be adequate.

TABLE 5.22: Summary of other clustering performance metrics for various clustering techniques.

	GMM	**K-means**	**Agglomerative**	**HDBSCAN**
Ajusted Rand Index	0.07-0.43	0.08-0.44	0.09-0.45	0.08-0.33
NMI	0.22-0.35	0.20-0.36	0.20-0.36	0.21-0.30
Accuracy	0.10-0.32	0.10-0.31	0.09-0.33	0.09-0.37

An automated marking system is evaluated based on how much time it can save a human marker whilst maintaining a certain standard of quality. Taking everything that has been discussed into consideration, Agglomerative clustering, where the number of components is set to 40, coupled with USE produces the best results.

It requires the minimum percentage of documents to be marked whilst achieving the necessary PC and CK metrics. It reaches the second highest maximum value and has the smallest range when looking at the average performance. The combination can save a marker approximately 65% of the time it would have normally taken to mark a question which is a significant saving.

There was no specific number of submissions to be marked per cluster which resulted in the minimum performance being met. Therefore, a system being implemented would need to actively calculate the performance metrics as more papers were successively marked.

5.6 The ECIC algorithm

The ECIC algorithm is implemented after clustering is performed in order to improve clusters that are formed. The algorithm works by identifying outliers in each cluster and reallocating them to a better suited cluster. As mentioned earlier, the data is split into a training set and test set where the test set is the outliers identified. It then uses logistic regression to classify the test set into the various clusters.

This process is supposed to be run a set number of times or till a certain convergence criteria is met. The algorithm is meant to maintain the number of clusters and improve the quality of clusters. The algorithm was implemented but did not run successfully through the predefined number of times. It identified the majority of datapoints as outliers and in the process eliminated most of the clusters.

As discussed earlier, the quality of the actual clusters is poor for various reasons. The ECIC algorithm has been implemented successfully but the data sets used in the original paper were significantly larger and the original quality of the clusters was much better with NMI values greater than 0.8 as seen in [28]. It is therefore shown that the ECIC algorithm does not work on datasets which only comprise of a few hundred data samples and where the clusters are very close to one another in the vector space.

Chapter 6

Conclusions and Future Work

6.1 Conclusion

An educator's role includes crucial components that are both tedious and essential, such as assessment and marking. According to Mason and Grove-Stephensen [15], 30% of a British teacher's time is dedicated to marking, while 40% is spent in the classroom. To test higher order thinking, free-text questions are created which are more difficult to mark given that they are open to interpretation. This research aimed to make steps in the automatic assessment of short-free text questions by developing an automatic short free-text marking system.

The research does this by using various sentence embedding methods such as USE, BERT and Doc2Vec to represent the text in a numerical way. Clustering methods such as GMM, Kmeans, Agglomerative clustering and UMAP coupled with HDBSCAN are implemented to group similar responses together. Various sampling strategies are then explored, whereby submissions are chosen to be graded.

It was found that Agglomerative clustering coupled with USE and using the sampling strategy whereby documents are marked based on their distance to the center of the cluster produced the best results in terms of finding a balance between time saved as well as meeting the performance criteria as mentioned by [3]. It was found that the combination could result in a question taking approximately 65% less time to be marked than it normally would.

Closer inspection of the quality of the clusters themselves showed poor performance in terms of what is generally expected from clustering algorithms however

there is no benchmark when it comes to solving this sort of problem. The poor performance is attributed to sparseness of the vectors and the proximity of the answers in the vector space, which creates difficulties in clustering them together. The ECIC algorithm, which is used to improve the quality of clusters, was implemented but did not work successfully on a dataset that only comprises of a few hundred samples.

6.2 Future Work

There are various ways in which this research could be expanded. Word embeddings could be used and a comparison between how sentence embeddings and words embeddings perform could be investigated. Other sampling techniques could also be explored. The ECIC algorithm could be modified to actually allow the number of clusters to change and to identify the impact that it has. Finally, active learning could be employed to determine which papers to grade that allow a system to reach a certain confidence interval.

Appendix A

GMM results

TABLE A.1: GMM results where minimum performance requirements are met

Dataset	Embed-ding	Comp-onents	Aimed marked	% marked	PC	CK	PC std	CK std
Stacks3	USE	40	5	55	0.700846	0.440865	0.027344	0.032252
		40	6	62	0.706587	0.443764	0.027288	0.035298
		40	7	67	0.709201	0.450187	0.024083	0.030078
		40	8	72	0.713295	0.449691	0.024745	0.029062
		40	9	76	0.71696	0.451914	0.024605	0.029101
Linked1	BERT	40	7	67	0.70269	0.421028	0.026219	0.022921
		40	8	72	0.705761	0.422035	0.02664	0.023846
		40	9	77	0.708109	0.420286	0.025992	0.024303
Queues2	BERT	20	9	62	0.700502	0.426538	0.023579	0.026243
		30	4	45	0.705968	0.432399	0.023491	0.023696
		30	5	54	0.714845	0.443057	0.022997	0.020324
		30	6	61	0.719487	0.446503	0.023494	0.021263
		30	7	68	0.721489	0.456061	0.023015	0.021955
		30	8	74	0.724209	0.466285	0.023268	0.018445
		30	9	79	0.728524	0.468465	0.020702	0.018386
		40	2	32	0.713296	0.451594	0.025172	0.025875
		40	3	45	0.728518	0.458229	0.024131	0.023817
		40	4	56	0.737203	0.466304	0.024891	0.02522
		40	5	65	0.743418	0.480589	0.024332	0.021245

Table A.1 continued from previous page

		40	6	70	0.747102	0.484174	0.023575	0.020429
		40	7	79	0.749046	0.490264	0.023781	0.019592
		40	8	84	0.751994	0.492093	0.024695	0.018485
		40	9	88	0.755311	0.493423	0.023011	0.019368
	USE	30	7	62	0.703419	0.416889	0.027409	0.031832
		30	8	68	0.707211	0.418773	0.024696	0.029107
		30	9	72	0.710537	0.421219	0.023947	0.028292
		40	2	31	0.702764	0.454124	0.027888	0.024535
		40	3	43	0.714989	0.451796	0.025445	0.024904
		40	4	52	0.727868	0.454571	0.026821	0.026426
		40	5	61	0.732548	0.457003	0.024522	0.02294
		40	6	67	0.740172	0.460992	0.022426	0.024869
		40	7	73	0.746149	0.466024	0.021339	0.023576
		40	8	78	0.749948	0.468447	0.022141	0.022104
		40	9	82	0.752377	0.468714	0.022455	0.02273
Trees1	BERT	40	5	76	0.720068	0.420544	0.023676	0.022706
		40	6	83	0.725404	0.41636	0.022974	0.02413
		40	7	88	0.730787	0.413866	0.023585	0.021626
		40	8	92	0.733284	0.415847	0.023556	0.0245
		40	9	95	0.734756	0.416972	0.023384	0.023257
	USE	40	3	46	0.713069	0.434521	0.032316	0.021749
		40	4	55	0.727357	0.434478	0.033735	0.019821
		40	5	64	0.731563	0.434829	0.035639	0.025166
		40	6	70	0.732714	0.434659	0.036493	0.023661
		40	7	76	0.735777	0.43732	0.035536	0.022524
		40	8	81	0.73993	0.443282	0.035033	0.021743
		40	9	85	0.742904	0.438564	0.033645	0.024914

Appendix B

K-means results

TABLE B.1: K-means results where minimum performance requirements are met

Dataset	Embedding	Components	Aimed marked	% marked	PC	CK	PC std	CK std
Stacks3	USE	40	5	54.031532	0.703298	0.440087	0.029308	0.026261
		40	6	60.135135	0.709053	0.444029	0.028735	0.027903
		40	7	65.608108	0.711787	0.452979	0.027737	0.027643
		40	8	70.472973	0.715336	0.451436	0.026113	0.025918
		40	9	74.617117	0.717745	0.454488	0.026883	0.024278
Linked1	BERT	40	6	60.403727	0.702267	0.428942	0.022366	0.028793
		40	7	67.158385	0.706375	0.425711	0.023942	0.025908
		40	8	72.872671	0.710380	0.424424	0.022919	0.021457
		40	9	77.701863	0.712852	0.422398	0.022662	0.020275
Queues2	BERT	20	8	56.717391	0.702078	0.426727	0.021962	0.026112
		20	9	61.891304	0.704144	0.430406	0.022619	0.027349
		30	3	35.891304	0.711332	0.442910	0.025192	0.020386
		30	4	45.521739	0.718392	0.442424	0.022071	0.020113
		30	5	54.217391	0.722765	0.454413	0.023569	0.019590
		30	6	61.630435	0.725458	0.463570	0.024879	0.020383
		30	7	68.173913	0.729880	0.470222	0.023493	0.022176
		30	8	73.608696	0.734645	0.477169	0.021355	0.023180
		30	9	78.347826	0.736250	0.479208	0.020468	0.024041
		40	2	32.500000	0.711331	0.448839	0.020917	0.020080

Table B.1 continued from previous page

		40	3	45.239130	0.728139	0.465093	0.018732	0.017862
		40	4	56.086957	0.738448	0.475066	0.020882	0.021663
		40	5	64.978261	0.743465	0.477658	0.021950	0.024014
		40	6	72.413043	0.747391	0.482746	0.020671	0.023895
		40	7	78.500000	0.751198	0.486418	0.020061	0.025459
		40	8	83.500000	0.755565	0.489240	0.019779	0.022746
		40	9	87.717391	0.758773	0.490879	0.019234	0.023712
	USE	30	4	41.869565	0.702713	0.426394	0.023603	0.020072
		30	5	49.652174	0.705587	0.427964	0.020590	0.026271
		30	6	56.478261	0.713041	0.433655	0.022080	0.025428
		30	7	62.434783	0.718547	0.438617	0.020787	0.021702
		30	8	67.434783	0.723309	0.439500	0.017478	0.019271
		30	9	71.891304	0.729633	0.442698	0.020160	0.018487
		40	2	31.173913	0.709584	0.446502	0.028991	0.034417
		40	3	42.086957	0.725476	0.449389	0.028279	0.030580
		40	4	51.673913	0.739019	0.457414	0.024722	0.026814
		40	5	60.195652	0.743081	0.460138	0.023748	0.027402
		40	6	67.086957	0.748214	0.469253	0.019348	0.025079
		40	7	72.826087	0.753267	0.471434	0.019906	0.027491
		40	8	77.521739	0.757781	0.477145	0.018791	0.026090
		40	9	81.391304	0.762024	0.478497	0.018355	0.021850
Trees1	BERT	40	3	54.043716	0.700601	0.422641	0.036301	0.022846
		40	4	66.448087	0.716508	0.421634	0.032803	0.026601
		40	5	76.229508	0.730370	0.425015	0.028804	0.025686
		40	6	82.923497	0.734971	0.428320	0.025810	0.022480
		40	7	88.005464	0.739036	0.424041	0.025947	0.026259
		40	8	91.748634	0.741833	0.420237	0.024110	0.023950
		40	9	94.453552	0.742804	0.416717	0.024244	0.025932
	USE	30	7	62.786885	0.701755	0.407443	0.036683	0.041549
		30	8	67.677596	0.704664	0.413775	0.037071	0.036558
		30	9	72.295082	0.706255	0.412574	0.038199	0.035325
		40	3	44.480874	0.718778	0.441383	0.029784	0.026797

Table B.1 continued from previous page

		40	4	53.715847	0.727338	0.436022	0.029786	0.027011
		40	5	61.639344	0.734783	0.440760	0.027739	0.025622
		40	6	68.442623	0.736523	0.437147	0.027408	0.022456
		40	7	73.934426	0.740494	0.437566	0.026740	0.024629
		40	8	78.825137	0.743980	0.437239	0.025083	0.022145
		40	9	83.114754	0.745736	0.439245	0.025914	0.020642

Appendix C

Hierarchical Agglomerative clustering results

TABLE C.1: Agglometric results where minimum performance requirements are met

Dataset	Embedding	Components	Aimed marked	% marked	PC	CK
Stacks3	USE	20	7	47.747748	0.700432	0.435674
		20	8	52.702703	0.703518	0.442217
		20	9	56.756757	0.708347	0.433265
		30	3	33.783784	0.701635	0.437813
		30	4	42.342342	0.710597	0.454694
		30	5	49.099099	0.722526	0.479391
		30	6	55.405405	0.723572	0.456727
		30	7	60.810811	0.727738	0.479754
		30	8	64.864865	0.728756	0.479754
		30	9	68.018018	0.732645	0.474538
		40	3	39.639640	0.720991	0.452112
		40	4	48.198198	0.730621	0.466229
		40	5	55.405405	0.729029	0.471805
		40	6	60.810811	0.733176	0.484002
		40	7	65.765766	0.744956	0.493624
		40	8	69.369369	0.749903	0.507165
		40	9	72.522523	0.750453	0.498304

Table C.1 continued from previous page

Linked1	BERT	30	9	67.701863	0.702526	0.419393
		40	8	73.291925	0.707540	0.427604
		40	9	78.260870	0.714347	0.430018
Queues2	BERT	20	7	49.130435	0.707721	0.413251
		20	8	53.913043	0.714330	0.443734
		20	9	58.695652	0.711451	0.439992
		30	2	26.086957	0.714956	0.475857
		30	3	37.391304	0.732669	0.483470
		30	4	46.956522	0.739771	0.474387
		30	5	55.217391	0.742901	0.469701
		30	6	62.173913	0.738116	0.475099
		30	7	68.695652	0.741655	0.465989
		30	8	73.478261	0.745183	0.476881
		30	9	77.826087	0.743706	0.482054
		40	2	33.478261	0.735357	0.493306
		40	3	46.956522	0.762013	0.515365
		40	4	57.391304	0.770968	0.505113
		40	5	66.086957	0.766520	0.493417
		40	6	73.478261	0.776139	0.511433
		40	7	79.565217	0.779482	0.520474
		40	8	84.347826	0.780584	0.515734
		40	9	88.260870	0.781700	0.517609
	USE	30	3	36.086957	0.703448	0.403382
		30	4	45.217391	0.719540	0.427485
		30	5	53.478261	0.713716	0.426276
		30	6	61.304348	0.716441	0.431902
		30	7	68.260870	0.727828	0.441305
		30	8	74.347826	0.730520	0.413273
		30	9	79.565217	0.732029	0.427090
		40	3	45.652174	0.738879	0.427948
		40	4	56.956522	0.751752	0.445318
		40	5	66.086957	0.743346	0.454594

Table C.1 continued from previous page

		40	6	73.478261	0.747102	0.464659
		40	7	79.565217	0.760735	0.474207
		40	8	83.478261	0.761070	0.466602
		40	9	86.956522	0.760596	0.466602
Trees1	BERT	40	3	56.830601	0.719653	0.411057
		40	4	69.398907	0.714546	0.427797
		40	5	78.688525	0.716787	0.409181
		40	6	85.245902	0.726710	0.419664
		40	7	90.710383	0.722706	0.418832
		40	8	95.628415	0.732416	0.418832
		40	9	98.360656	0.736875	0.415677
	USE	30	7	71.038251	0.730219	0.435384
		30	8	77.049180	0.733525	0.431308
		30	9	80.874317	0.733154	0.436248
		40	2	37.158470	0.737193	0.453334
		40	3	48.087432	0.742042	0.463343
		40	4	59.016393	0.749248	0.464747
		40	5	67.213115	0.766599	0.471604
		40	6	73.770492	0.765755	0.468842
		40	7	79.234973	0.768425	0.466312
		40	8	83.606557	0.772686	0.466312
		40	9	86.885246	0.772661	0.468836

Appendix D

HDBSCAN results

TABLE D.1: HDBSCAN results where minimum performance requirements are met

Dataset	**Embedding**	min_cluster	**Components**	**Aimed marked**	**% marked**	**PC**	**CK**
Stacks3	USE	2	38	2	34	0,705497	0,525467
		2	38	3	50	0,749269	0,527239
		2	38	4	62	0,770165	0,549727
		2	38	5	70	0,77842	0,555632
		2	38	6	76	0,779807	0,55047
		2	38	7	80	0,77975	0,541626
		2	38	8	83	0,780905	0,539388
		2	38	9	84	0,780892	0,539388
		3	22	3	30	0,740898	0,493246
		3	22	4	39	0,749074	0,51438
		3	22	5	47	0,748682	0,518281
		3	22	6	53	0,750917	0,535205
		3	22	7	58	0,75051	0,521228
		3	22	8	62	0,750357	0,525964
		3	22	9	65	0,756303	0,530559
Linked1	BERT	2	54	2	34	0,726128	0,47042
		2	54	3	49	0,748671	0,498517
		2	54	4	60	0,771697	0,501015
		2	54	5	68	0,782491	0,505482

Table D.1 continued from previous page

		2	54	6	75	0,783786	0,505354
		2	54	7	79	0,786125	0,509941
		2	54	8	83	0,788073	0,507902
		2	54	9	85	0,787072	0,51102
		3	27	3	25	0,717128	0,466478
		3	27	4	34	0,753496	0,472663
		3	27	5	41	0,75674	0,474372
		3	27	6	49	0,756913	0,479534
		3	27	7	56	0,754635	0,477777
		3	27	8	61	0,759517	0,475931
		3	27	9	66	0,763492	0,475295
		4	24	3	22	0,739585	0,501007
		4	24	4	30	0,752058	0,501683
		4	24	5	37	0,755162	0,491222
		4	24	6	45	0,76216	0,482521
		4	24	7	51	0,763003	0,494313
		4	24	8	57	0,767117	0,489521
		4	24	9	63	0,770551	0,493939
	USE	2	59	5	74	0,703143	0,451643
		2	59	6	79	0,708385	0,456064
		2	59	7	83	0,711526	0,453849
		2	59	8	87	0,715287	0,460761
		2	59	9	89	0,718389	0,460761
		5	20	9	52	0,701604	0,475054
Queues2	BERT	2	37	4	59	0,711303	0,435078
		2	37	5	67	0,717759	0,459025
		2	37	6	73	0,72822	0,461295
		2	37	7	79	0,734684	0,487734
		2	37	8	82	0,738395	0,485927
		2	37	9	84	0,742318	0,478763
		5	14	4	24	0,704982	0,493529
		5	14	5	30	0,703003	0,488115

Table D.1 continued from previous page

		5	14	6	37	0,70316	0,485693
		5	14	7	42	0,710716	0,514819
		5	14	8	47	0,709321	0,495327
		5	14	9	50	0,714176	0,495327
	USE	2	33	4	53	0,717403	0,458305
		2	33	5	61	0,731903	0,470414
		2	33	6	67	0,739212	0,466724
		2	33	7	71	0,752224	0,485548
		2	33	8	75	0,754789	0,484415
		2	33	9	77	0,757261	0,476407
		3	18	7	49	0,706774	0,435164
		3	18	8	54	0,709339	0,452495
		3	18	9	58	0,7067	0,440772
		4	17	5	36	0,700156	0,449531
		4	17	6	40	0,705429	0,44408
		4	17	7	45	0,714038	0,472966
		4	17	8	50	0,719107	0,470687
		4	17	9	54	0,716586	0,472477
		5	11	7	33	0,705877	0,459969
		5	11	8	38	0,700563	0,4502
		5	11	9	43	0,701964	0,444822
Trees1	BERT	2	34	2	37	0,740823	0,480035
		2	34	3	55	0,794889	0,475522
		2	34	4	69	0,798431	0,492361
		2	34	5	79	0,806768	0,500402
		2	34	6	84	0,81129	0,495252
		2	34	7	87	0,811803	0,495252
		2	34	8	88	0,811817	0,495252
		2	34	9	89	0,811845	0,495252
		3	20	6	60	0,743575	0,415348

Bibliography

[1] Dimo Angelov. "Top2vec: Distributed representations of topics". In: *arXiv preprint arXiv:2008.09470* (2020).

[2] Somnath Banerjee, Krishnan Ramanathan, and Ajay Gupta. "Clustering short texts using wikipedia". In: *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. 2007, pp. 787–788.

[3] Gavin Brown. "The reliability of essay scores: The necessity of rubrics and moderation". In: *Tertiary assessment and higher education student outcomes: Policy, practice and research* (2009), pp. 40–48.

[4] U Cambridge. *Online edition (c) 2009 Cambridge UP An Introduction to Information Retrieval Christopher D*. 2009.

[5] Ricardo JGB Campello et al. "Hierarchical density estimates for data clustering, visualization, and outlier detection". In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 10.1 (2015), pp. 1–51.

[6] Daniel Cer et al. "Universal sentence encoder". In: *arXiv preprint arXiv:1803.11175* (2018).

[7] Xueqi Cheng et al. "Btm: Topic modeling over short texts". In: *IEEE Transactions on Knowledge and Data Engineering* 26.12 (2014), pp. 2928–2941.

[8] Jacob Cohen. "A coefficient of agreement for nominal scales". In: *Educational and psychological measurement* 20.1 (1960), pp. 37–46.

[9] Alexis Conneau et al. "Supervised learning of universal sentence representations from natural language inference data". In: *arXiv preprint arXiv:1705.02364* (2017).

[10] Jacob Devlin et al. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[11] Fei Dong, Yue Zhang, and Jie Yang. "Attention-based recurrent convolutional neural network for automatic essay scoring". In: *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*. 2017, pp. 153–162.

[12] Hugging Face. *"bert-base-nli-mean-tokens"*. URL: "https://huggingface.co/sentence-transformers/bert-base-nli-mean-tokens".

[13] Maarten Grootendorst. "BERTopic: Neural topic modeling with a class-based TF-IDF procedure". In: *arXiv preprint arXiv:2203.05794* (2022).

[14] Amir Hadifar et al. "A self-training approach for short text clustering". In: *Proceedings of the 4th Workshop on Representation Learning for NLP (RepL4NLP-2019)*. 2019, pp. 194–199.

[15] Trevor Hastie et al. *The elements of statistical learning: data mining, inference, and prediction*. Vol. 2. Springer, 2009.

[16] Xiaofei He et al. "Laplacian regularized gaussian mixture model for data clustering". In: *IEEE Transactions on Knowledge and Data Engineering* 23.9 (2010), pp. 1406–1418.

[17] Jerry A Jacobs and Sarah E Winslow. "The academic life course, time pressures and gender inequality". In: *Community, Work & Family* 7.2 (2004), pp. 143–161.

[18] David R Krathwohl. "A revision of Bloom's taxonomy: An overview". In: *Theory into practice* 41.4 (2002), pp. 212–218.

[19] Quoc Le and Tomas Mikolov. "Distributed representations of sentences and documents". In: *International conference on machine learning*. PMLR. 2014, pp. 1188–1196.

[20] Yinhan Liu et al. "Roberta: A robustly optimized bert pretraining approach". In: *arXiv preprint arXiv:1907.11692* (2019).

[21] J MacQueen. "Classification and analysis of multivariate observations". In: *5th Berkeley Symp. Math. Statist. Probability*. University of California Los Angeles LA USA. 1967, pp. 281–297.

[22] Oliver Mason and Ian Grove-Stephensen. "Automated free text marking with paperless school". In: (2002).

[23] Leland McInnes, John Healy, and James Melville. "Umap: Uniform manifold approximation and projection for dimension reduction". In: *arXiv preprint arXiv:1802.03426* (2018).

[24] Tomas Mikolov et al. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[25] Michael Mohler and Rada Mihalcea. "Text-to-text semantic similarity for automatic short answer grading". In: *Proceedings of the 12th Conference of the European Chapter of the ACL (EACL 2009)*. 2009, pp. 567–575.

[26] Karl Pearson. "VII. Note on regression and inheritance in the case of two parents". In: *proceedings of the royal society of London* 58.347-352 (1895), pp. 240–242.

[27] Leonid Pugachev and Mikhail Burtsev. "Short Text Clustering with Transformers". In: *arXiv preprint arXiv:2102.00541* (2021).

[28] Md Rashadul Hasan Rakib et al. "Enhancement of Short Text Clustering by Iterative Classification". In: *International Conference on Applications of Natural Language to Information Systems*. Springer. 2020, pp. 105–117.

[29] Nils Reimers and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks". In: *arXiv preprint arXiv:1908.10084* (2019).

[30] D. Sarkar. *Implementing Deep Learning Methods and Feature Engineering for Text Data: The Skip-gram Model*. URL: https://www.kdnuggets.com/2018/04/implementing-deep-learning-methods-feature-engineering-text-data-skip-gram.html.

[31] Raheel Siddiqi, Christopher J Harrison, and Rosheena Siddiqi. "Improving teaching and learning through automated short-answer marking". In: *IEEE Transactions on Learning Technologies* 3.3 (2010), pp. 237–249.

[32] Alexander Strehl and Joydeep Ghosh. "Cluster ensembles—a knowledge reuse framework for combining multiple partitions". In: *Journal of machine learning research* 3.Dec (2002), pp. 583–617.

[33] Nguyen Xuan Vinh, Julien Epps, and James Bailey. "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" In: *Proceedings of the 26th annual international conference on machine learning*. 2009, pp. 1073–1080.

[34] Jiaming Xu et al. “Self-taught convolutional neural networks for short text clustering”. In: *Neural Networks* 88 (2017), pp. 22–31.

[35] Jianhua Yin and Jianyong Wang. “A model-based approach for text clustering with outlier detection”. In: *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE. 2016, pp. 625–636.

[36] Chu Tao Zheng, Cheng Liu, and Hau San Wong. “Corpus-based topic diffusion for short text clustering”. In: *Neurocomputing* 275 (2018), pp. 2444–2458.