Eötvös Loránd University
Faculty of Informatics

Enhancing the usability of Automatic Essay Evaluation

Tashu,Tsegaye Misikir
Data Science and Engineering Department

Mariya Saprikina
Computer Science

December 2019, Budapest

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Automatic Essay Evaluation (AEE) systems were introduced to alleviate the workload of the assessors and to improve the feedback cycle in the teaching-learning process in the context of large-scale assessment. Since its introduction, several research activities have been carried out and Many analytical researches have introduced relatively high correlation between human scores and the scores produced by AEE. The task of AEE was regarded as a machine learning problem that learns to approximate the assessment process using handcrafted features with supervised machine learning approaches. Most of the state-of-the-art AEE systems rely on supervised machine learning approaches that require a large number of manually annotated training sets to train the scoring engine and make the scoring engine work well. There are cases in which it is difficult to find labeled training short answers that require a great deal of effort when creating labeled sets.

Every essay input has to pass through a proper validation mechanism before computing the score to increase the use-ability of AEE systems. Every well-written essay that does not address the question topic may receive a good score from an AEE system because of linguistic features, such as text structure and surface [42]. If every essay submitted to the AEE system is evaluated as a standard essay input, this may degrade user confidence in the AEE engine. Such essay might be either off-topic or might be plagiarized.

The main objective of our research is to increase the use-ability of automatic

essay evaluation system by addressing the issues described above. Therefore, the main objective of our research are the following:

1. To design an approach Off-topic essay detection for identification of student essay which is not written to the test question topic.
2. To design an approach that can be used to detect copied student solutions and train the model to distinct various levels of plagiarism.
3. To introduce an efficient approach for selecting small-sized set to be annotated by human rater automatically that will be used to score the majority of essay using semantic k-means.

1.2 Thesis outline

The structure of the thesis is described as follows:

- **Chapter 2** provides a theoretical overview about the input representation techniques and the models we implemented in the research.
- **Chapter 3** provides a detailed explanation about the datasets and development tools required for the experiments, and also a thorough review of the results that were obtained during the implementations.
- **Chapter 4** represents a conclusion of the work and outlines future research activities.

Chapter 2

Theoretical review

2.1 Data preprocessing

The first step that needs to be done in any of machine learning tasks is data preprocessing. The problem is that machines are not able to extract meaningful information from raw and unstructured data. Here come some natural language processing (NLP) techniques that deal with such kind of data and converts raw texts into cleaned tokens that can be readable by machine learning algorithms and be used for different semantic analysis methods.

2.1.1 Tokenization

Tokenization is a process of breaking strings into tokens. These tokens can be represented as words, sentences, characters or punctuation symbols. It is a very important step in natural language processing and should be done before any other preprocessing step.

When tokenizing the document, it is important to be aware that not every token is valuable. Some web links, email addresses and other unwanted characters may appear. There are some techniques in natural language processing that help to remove this kind of tokens. In our text preprocessing, we used the following techniques:

1. Punctuation removal - removes all unnecessary punctuation marks.

2. Stopwords removal - removes the most common words in natural language. There are 179 English words in nltk library, including ‘i’, ‘me’, ‘my’, ‘myself’,

'we', 'you', 'he', 'his' and etc. All of them have no contribution to deriving semantics from texts. However, if a corpus is small, the removing stopwords would decrease the total number of words by a large percent.

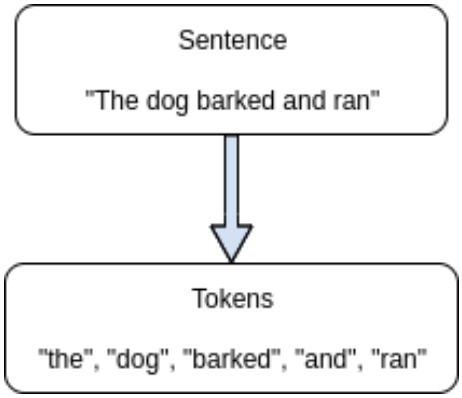


Figure 2.1: Example of tokenization

2.1.2 Stemming and lemmatization

In human language, some words may have different forms which are derived from the same root but have a similar meaning. For example, "work" and "worked" mean the same thing. However, in natural language processing, these two words will be recognized as words with totally different meanings. Methods like Stemming and Lemmatization were suggested and used to address the problems stated.

These methods are widely used in Search Engine Optimizations (SEOs), tagging systems, indexing and information retrieval. For instance, when searching for "friend" on Google, it will result in "friends", "friendship" because "friend" is the root for each of them.

Stemming is a process of mapping a group of words to the same stem even if the stem does not exist in natural language. So stemming may produce the words that are not used by humans in that particular form.

Original word	Stemmed word
double	doubl
doubled	doubl
doubling	doubl
doubles	doubl

Figure 2.2: Example of stemming

Lemmatization method is very similar to stemming. The difference is that unlike stemming, it cuts the word ensuring that the root belongs to the human language. In lemmatization, the root word is called Lemma. A lemma is a dictionary form of a set of words.

Original word	Lemmatized word
double	double
doubled	double
doubling	double
doubles	double

Figure 2.3: Example of lemmatization

2.2 Feature Extraction - Bag-of-words model

As it has already been mentioned above, machine learning algorithms are not able to work with raw texts; hence, the text documents have to be converted into vectors of numbers. This concept is called feature extraction. One of the popular methods of this concept is the bag-of-words (BoW) model. The earliest reference to "bag of words" can be found in [17]. The model represents a text as the occurrences or frequency of terms (words) in each document. Each document will be described as a vector of real numbers where each number corresponds to the frequency of a particular term in the dictionary. Let's suppose we have a corpus $D = \{d_1, d_2, ..., d_n\}$ and the dictionary $T = \{t_1, t_2, ..., t_m\}$. Then the output of representation will be an

$\mathbb{R}^{n \times m}$ matrix called the document matrix.

$$\begin{array}{c} \\ term_1 \\ term_2 \\ \vdots \\ term_m \end{array} \begin{array}{c} \begin{array}{cccc} document_1 & document_2 & \dots & document_n \end{array} \\ \begin{pmatrix} w(t_1,d_1) & w(t_2,d_2) & \dots & w(t_1,d_n) \\ w(t_2,d_1) & w(t_2,d_2) & \dots & w(t_2,d_n) \\ \vdots & \vdots & \ddots & \vdots \\ w(t_m,d_1) & w(t_m,d_2) & \dots & w(t_m,d_n) \end{pmatrix} \end{array}$$

Figure 2.4: Term-document matrix

	Document 1	Document 2	Document 3
weather	35	4	8
summer	0	23	11
breeze	3	8	0
storm	10	0	20

Figure 2.5: Example of Bag-of-Words

2.3 Term Frequency-Inverse Document Frequency (TF-IDF)

2.3.1 Term Frequency (TF)

Assume we have a corpus of English documents and would like to fetch the most relevant ones to the query "the white dog". The simplest way is to filter out the documents which do not contain such words at all. Nevertheless, that is not enough because there will also be chosen the documents where their words might occur just once. Therefore, we might consider that the more times the word appears in the document, the more relevant it is for the given query. This approach is called *term frequency*. However, if the corpus contains the documents with different lengths, we might need to adjust the frequency following the length of each document.
The weight of a term that occurs in a document is simply proportional to the term frequency[27]. Here comes the concept of *term weighting*.

Let $tf_{t,d}$ be the frequency of a term t in a document d.
There are several possible representations of term weighting:

- **Boolean frequency:** $tf_{t,d} = 1$ if t occurs in d and 0 otherwise.

$$w(t,d) = \begin{cases} 0, & t \notin d \\ 1, & t \in d \end{cases} \tag{2.1}$$

  It is important to note, that in this method the magnitude of each term to the document will be completely lost, as we ignore the number of times each term appears in the document.

- **Term frequency adjusted for document length:**

$$w(t,d) = \frac{tf_{t,d}}{\sum\limits_{t' \in d} tf_{t',d}}, \tag{2.2}$$

  where $\sum_{t' \in d} tf_{t',d}$ is the number of terms in d

- **Logarithmically scaled frequency:** soothes the effect of very frequent terms in a document.

$$w(t,d) = 1 + \log tf_{t,d} \tag{2.3}$$

2.3.2 Inverse document-frequency (IDF)

The problem with the previous example is that the word "the" is very common for most of the documents, hence, besides the documents containing words "white" and "dog", we will get all the documents containing the word "the" as well, which will significantly lower the accuracy of the document relevance to the given query. As a solution, a *inverse document frequency* concept was introduced [21] which emphasise the significance of rare words in corpora and, on the contrary, ignores very frequent words.

Considering df_t as the document frequency of term t in the entire corpus, the inverse document frequency is:

$$idf_t = \log \frac{N}{df_t}, \tag{2.4}$$

where N is the total number of documents in the corpus $N = |D|$.

We can also modify the idf by adding a parameter α which soothe the impact of very uncommon words.

$$idf_t = \log \frac{\alpha + N}{\alpha + df_t} \tag{2.5}$$

2.3.3 TF-IDF

Summarizing the concepts discussed above, we can now calculate the tf-idf using the combination of term frequency and the inverse document frequency. [36]

$$tfidf_{d,t} = tf_{d,t} \times idf_t \tag{2.6}$$

There are many options for measuring tf-idf, and some of them might give quite good results depending on the given corpora, as shown in [35]. For instance, calculating tf-idf using a logarithmically scaled term frequency (Formula 2.3) will significantly lower the weight given to common terms. It could be useful when term frequencies follow a power law with respect to the rank.

2.3.4 Limitations of TF-IDF approach

TF-IDF is based on Bag-of-Words model; therefore, it ignores such properties as the position of terms in documents. For example, "This is a bicycle, not a car" and "This is a car, not a bicycle" will have the identical vector representation, although the meaning is totally different. This approach also does not capture the semantic similarity between documents compared to word embedding, topic modelling and others will be covered later. Considering all these limitations, TF-IDF can only be useful as a lexical level concept.

2.4 Distributional semantics

As was already mentioned above, previous models of term vector representations have nothing to do with the semantic similarity between linguistic items of language data. Consider the following two sentences: "The weather is good" and "The weather

is great". They are linguistically very similar to each other. Let us try to analyse word vector representation of similar words "good" and "great" based on the existence of the word in the vocabulary. The vocabulary is [The,weather,is,good,great]. The one-hot encoded vectors will be represented as:

Good: [0,0,0,1,0];

Great: [0,0,0,0,1].

As these two vectors are orthogonal, the cosine similarity between them will be very low, although they have a similar meaning.

Another problem is that if we have a large corpus of, for example, millions of words, all the word vectors will be very sparse and, hence, not efficient. The tf-idf approach helps to reduce the dimensionality of vectors, yet it is not sufficient.

Distributional semantics is the research area which is focused on solving the above-mentioned problems, precisely, quantifying semantic similarities between words based on their distributional vector representations in large samples of language data.

The idea of a distributed representation of words has its roots in [15] as *a word is characterized by the company it keeps* which means that the words which are usually used in the same contexts tend to have similar meanings. Over the years, there was a significant contribution to the research about word embedding and dimensionality reduction of word vectors in [5, 13, 23, 33].

2.4.1 Classic Word Embedding Model

The term *word embedding* was originally mentioned by [5] in 2003 who trained the embeddings in a neural language model together with the model's parameters.

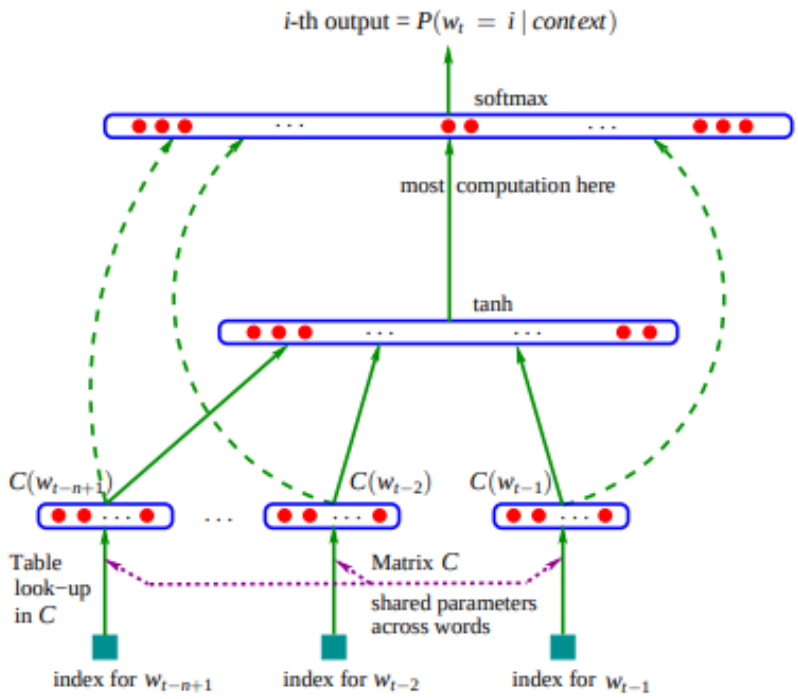


Figure 2.6: Neural language model [5]

The model shown in Figure 2.6 represents a neural architecture $f(i, w_{t-1}, \ldots, w_{t-n+1}) = g(i, C(w_{t-1}), \ldots, C(w_{t-n+1}))$ where g is the neural network and $C(i)$ is the i-th word feature vector. This model consists of one-hidden layer feed-forward neural network that is trained to predict the next word in the context. It takes a word from a vocabulary as input and embeds it as a vector into a lower-dimensional space, which it then fine-tunes through back-propagation, necessarily yields word embeddings as the weights of the first layer, which is usually referred to as *Embedding Layer*.

Let us assume we have a training corpus T with the sequence of words in it $w_1, w_2, w_3, ..., w_T$ that are taken from vocabulary V with a size of $|V$. The neural network model consists of n words. We associate every word with an input embedding v_w (the eponymous word embedding in the Embedding Layer) with d dimensions and an output embedding v'_w. We finally optimize an objective function J_θ with regard to our model parameters θ and our model outputs some score $f_\theta(x)$ for every input x.

$$J_\theta = \frac{1}{T} \sum_{t=1}^{T} \log f(w_t, w_{t-1}, ..., w_{t-n+1}) \tag{2.7}$$

$f(w_t, w_{t-1}, ..., w_{t-n+1})$ is the output of the model, i.e the probability

$p(w_t|w_{t-1}, ..., w_{t-n+1})$ as computed by the softmax, where n is the number of previous words fed into the model.

There are 3 layers used in the above neural language model:

1. Embedding Layer: This layer generates word embeddings by multiplying an index vector with a word embedding matrix;

2. Intermediate Layer(s): One or more layers that produce an intermediate representation of the input, e.g. a fully-connected layer that applies a non-linearity to the concatenation of word embeddings of n previous words;

3. Softmax Layer: The final layer that produces a probability distribution over words in V.

2.4.2 C&W neural language model

Later in 2008, an improved neural language model was trained on a larger dataset, and the computed word embeddings were introduced as a highly effective tool when used in downstream tasks. [13]

In order to avoid computing the expensive softmax, they came up with employing an alternative objective function: rather than the cross-entropy criterion of [5], which maximizes the probability of the next word given the previous words, [13] train a network to output a higher score f_θ for a correct word sequence (a probable word sequence in [5]) than for an incorrect one. For this purpose, they use a pairwise ranking criterion, which is the following:

$$J_\theta = \sum_{x \in X} \sum_{w \in V} max\{0, 1 - f_\theta(x) + f_\theta(x^{(w)})\} \tag{2.8}$$

They sample correct windows x containing n words from the set of all possible windows X in their corpus. For each window x, they then produce a corrupted, incorrect version $x(w)$ by replacing x's centre word with another word w from V. Their objective now maximises the distance between the scores output by the model for the correct and the incorrect window with a margin of 1.

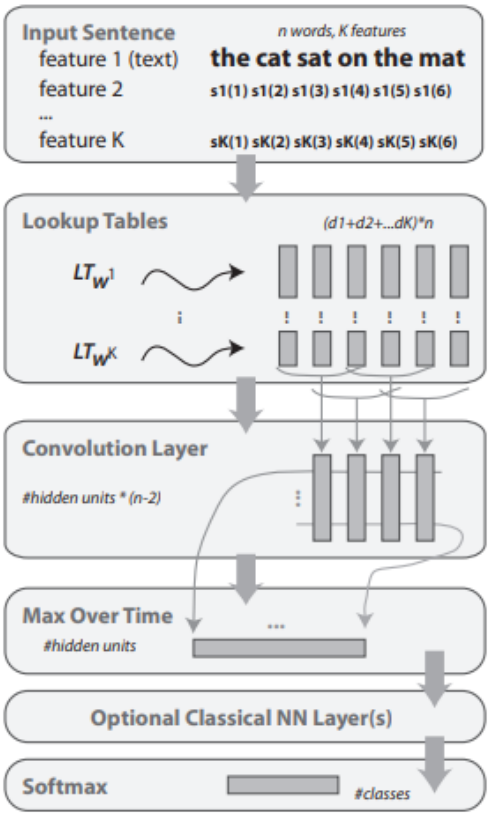


Figure 2.7: A C&W neural language model [13]

2.4.3 Word2Vec

However, the most effective way of training word embeddings was introduced by [29] in 2013. They created a word embedding toolkit called *word2vec* which can train vector space models much faster than it previously was.

Tomas Mikolov [29] introduced two architectures to learn word embeddings (Continuous bag-of-words and Skip-gram) which turned out to be a way less computationally expensive compared to the previous neural language models.

- **Continuous bag-of-words (CBOW)**

  In contrast to the neural language model that is only able to make predictions on past words [5, 13], this approach uses both n words before and after the target word w_t for this purpose (Figure 2.8). The model is called *continuous*

bag-of-words to underline that the order of words is not preserved as well as in the classic bag-of-words method.

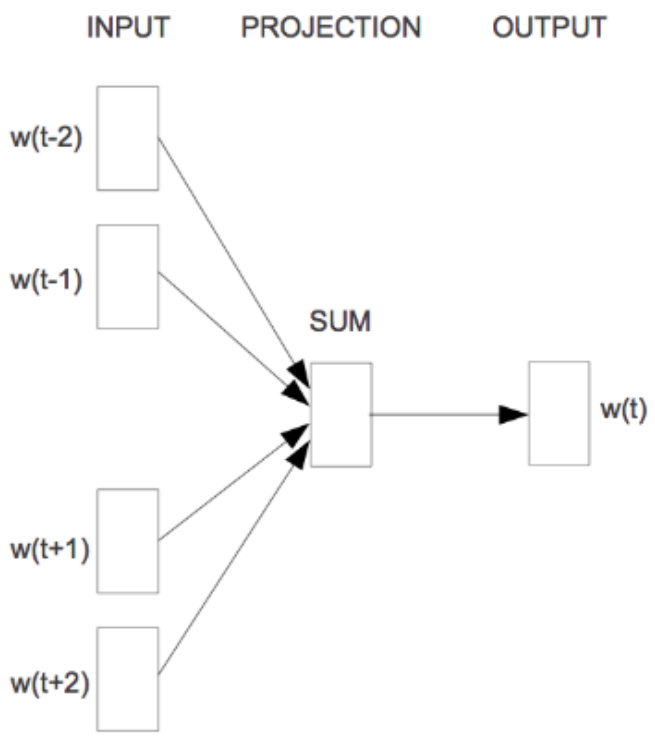


Figure 2.8: CBOW model [29]

The math representation of the above scheme is the following:

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T} \log p(w_t, w_{t-1}, w_{t+1}, ..., w_{t+n}) \tag{2.9}$$

This model receives a window of n context words around a target word w_t at each time step t instead of feeding n past words into the model.

- **Skip-gram**

  Unlike CBOW model which predicts the target word using the context words, Skip-gram uses the target word as an input to predict the context words the target word is surrounded by (Figure 2.9).

  As shown in the Formula 2.10, skip-gram sums the log probabilities of n context words to the left and right of the target word w_t:

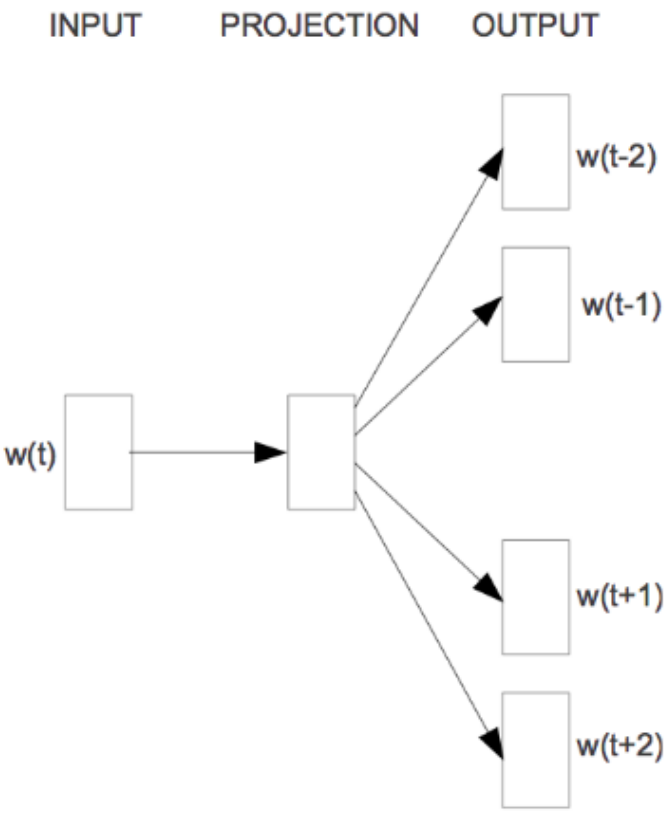


Figure 2.9: Skip-gram model [29]

$$J_\theta = \frac{1}{T}\sum_{t=1}^{T}\sum_{-n\leq j\leq n,\neq 0} \log p(w_{t+j}|w_t) \tag{2.10}$$

Once the word2vec model training converges, we result with the distributed word embeddings which have a very high correlation with semantic regularities in human language. For example, "King - Man + Woman" will result in a vector which is very closed to the vector of a word "Queen" [30].

2.4.4 Doc2Vec

In our experiments, we are dealing with sets of documents; hence, we were looking for document vector representation techniques which perform in the most effective way. In this chapter, we have already covered a classic Bag-of-Words model and TF-IDF. However, both techniques are not always useful due to such limitations as not preserving the word order in the context and not capturing the semantic similarity

between words and their corresponding contexts.

One of the most recent and popular is *Doc2Vec*. This model is an actual extension of a previously discussed framework called Word2Vec. The goal of Doc2Vec is to determine a continuous paragraph or a document vector in order to preserve the semantic similarity among a set of documents [24].

- **Paragraph vector: A distributed memory model. (PV-DM)**

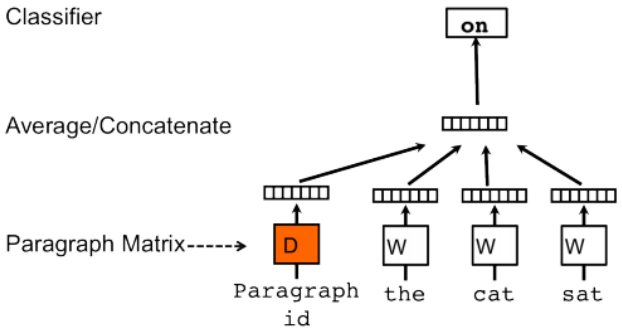


Figure 2.10: PV-DM framework. [24]

This model is very similar to the CBOW Word2Vec model. The only extension that was made is adding the Paragraph Vector. Each paragraph in a document is mapped to a unique vector represented by a column in matrix D. Each word in its turn is also mapped to a unique vector represented by a column in matrix W. The paragraph vector and word vectors are concatenated to predict the next word given the context. In other words, the paragraph vector plays the role of memory, remembering what was missing from the given context.

This method addresses two main limitations of a classic Bag-of-Words model. Firstly, it captures the semantic relationship between the words. Secondly, the word order is now taken into consideration. It helps to preserve much information, at least in a small context. This is a way like an n-gram model works. However, unlike paragraph vector model, the n-grams would create a very high-dimensional vector representation which is not efficient in case of having large corpus.

- **Paragraph Vector without word ordering: Distributed bag of words (DBOW)**

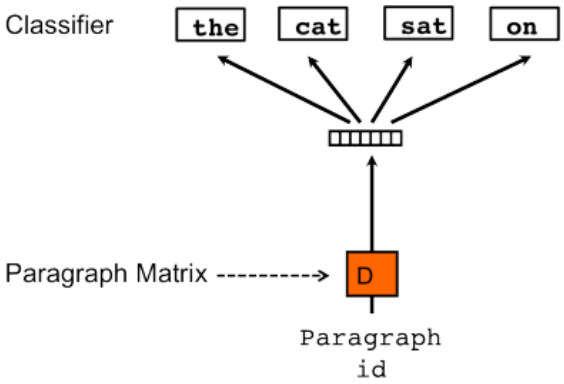


Figure 2.11: DBOW framework. [24]

As opposed to PV-DM method, the Paragraph vector in this method is trained to predict a small context of words which is quite similar to Skip-gram Word2Vec model. If the above Paragraph method predicts the next word in the context, this method ignores the context words in the input but makes the model predict the words randomly chosen from the paragraph in the output. This method also does not preserve the order of the context words and the model requires to store fewer data compared to the previous Paragraph method.

2.5 Models

This section describes the models we implemented to deal with three main problems of Automatic Essay Evaluation. In our methods, we used the following models: Latent Dirichlet Allocation (LDA), Latent Semantic Analysis (LSA), Word Mover's Distance (WMD) and K-means clustering.

2.5.1 Latent Dirichlet Allocation (LDA)

One of the primary goals in natural language processing is to analyse texts by its topics. The process of extracting such topics from a collection of documents is called *topic modeling*. A topic can be described as a set of words that usually occur in similar contexts. Hence, the main objective of topic modelling is to discover hidden or *latent* structure from a collection of documents that share the same content.

Latent Dirichlet Allocation is one of the most popular techniques in topic mod-

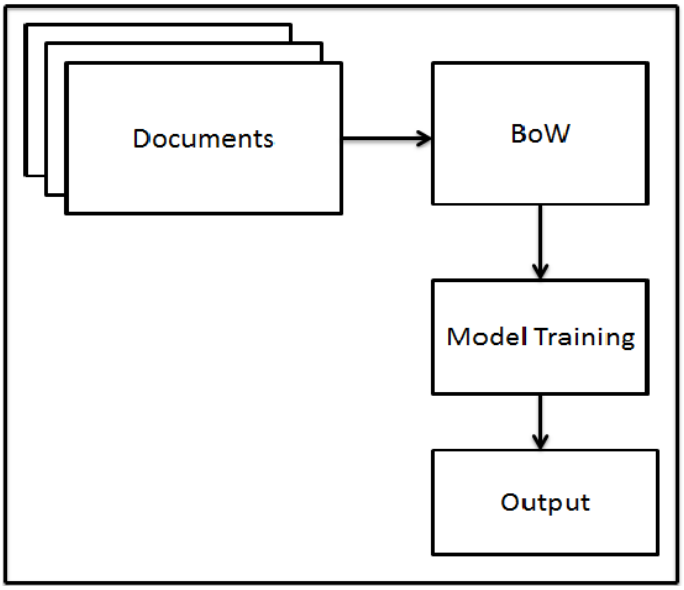


Figure 2.12: Topic modeling framework. [3]

eling. It treats the documents as a mixture of topics; these topics further generate words based of their probability distribution [9].

In LDA, we assume that there are k underlying latent topics according to which documents are generated, and that each topic is represented as a multinomial distribution over the $|V|$ words in the vocabulary. A document is generated by sampling a mixture of these topics and then sampling words from that mixture [8].

In LDA the terms are defined as:

- A word is a unit-basis vector fetched from a vocabulary $\{1, \ldots, V\}$
- A document is a sequence of N words denoted by $W = \{w_1, w_2, \ldots, w_N\}$
- A corpus is a collection of M documents denoted by $D = \{w_1, w_2, \ldots, w_M\}$.

The LDA defines the following generative process for each document W in a corpus D:

1. Choose $N \sim$ Poisson(ξ), where N is the length of the document
2. Choose $\theta \sim$Dir(α), where θ_i is the multinomial topic distribution for document i (a k-vector lies in the $(k-1)-$ simplex if $\theta_i \geq 0, \sum_{t=0}^{k} \theta_i = 1$) and α is a k-vector (k denotes the number of topics) with components $\alpha_k > 0$. The probability density function of the Dirichlet distribution is the following:

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\Pi_{i=1}^{k} \Gamma(\alpha_i)} \theta_1^{\alpha_i - 1} \cdots \theta_k^{\alpha_k - 1} \tag{2.11}$$

3. For each of the N words w_n:

   - Choose a topic $z_n \sim$ Multinomial(θ);
   - Choose a word w_n from a multinomial probability $p(w_n|z_n, \beta)$ conditioned on the topic z_n. The word probabilities are set by a $k \times V$ matrix β where $\beta_{ij} = p(w_j = 1|z_i = 1)$;
   - Note: each topic has a different probability of generating each word

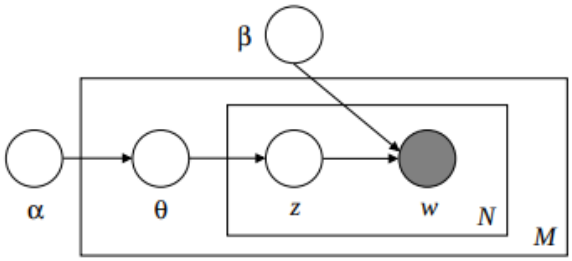


Figure 2.13: Graphical model representation of LDA. [9]

In LDA we generate both word distribution for each topic β_k and topic distribution for each document θ_d using Dirichlet distributions. For instance, by learning a k-vector α_i, we learn the topics distribution the document may have. The higher the α_i value is relevant to others, the more likely we will pick that topic.
In a Dirichlet distribution, the probability distribution is a sampled value from p which always sums up to one. This is a reason why Dirichlet distribution is called a *distribution of distributions*.

2.5.2 Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a technique in distributional semantics which analyze relationships between a set of documents and the terms they contain by producing a set of concepts related to the documents and terms. Similarly to LDA model, the main idea of LSA is that words with a close meaning will occur in similar

contexts of documents. It extracts similarities between documents using dimensionality reduction. Firstly, a classic term-document matrix is constructed from a large set of documents using Count Vectorizer (classic Bag-of-words) or Tf-Idf. Then, a popular mathematical technique called *singular value decomposition* (SVD) is applied to this matrix to reduce the number of rows without distorting the similarity structure among columns.

The latent semantic structure was originally introduced in [16] as a Latent Semantic Indexing (LSI). Both LSI and LSA follow the same method of reducing dimension, however, LSI is mostly used in the context of web search, while LSA works with large texts in natural language processing. [4] introduced significant improvements in speech recognition tasks due to the power of LSA to capture long-term context of text.

The basic algorithm of LSI/LSA techniques can be described as follows [10]:

1. A matrix A is formed, where in each row corresponds to a term that appears in the documents, and each column corresponds to a document. Each element $a_{m,n}$ in the matrix corresponds to the number of times that the term m occurs in document n.

2. Local and global term weighting is applied to the entries in the term-document matrix. This weighting may be applied in order to achieve multiple objectives, including compensating for differing lengths of documents and improving the ability to distinguish among documents. Some very common words such as *and, the*, etc. typically are deleted entirely (i.e., treated as stopwords).

3. Singular value decomposition (SVD) is used to reduce this matrix to a product of three matrices:

$$A = U\Sigma V^T \tag{2.12}$$

   Where:

   - $A \in \mathbb{R}^{t \times d}$ - term-document matrix
   - $U \in \mathbb{R}^{t \times t}$ - term-term orthogonal matrix having the left singular vectors of A as columns

- $V \in \mathbb{R}^{d \times d}$ - document-document orthogonal matrix having the right singular values of A as columns
- $\Sigma \in \mathbb{R}^{d \times d}$ - term-document diagonal matrix whose elements are the singular values of A (the non-negative square roots of the eigenvalues of AA^T)

4. Dimensionality is reduced by deleting all but the k largest values of Σ, together with the corresponding columns in U and V, yielding an approximation of A:

$$A_k = U_k \Sigma_k V_k^T \tag{2.13}$$

which is the best rank-k approximation to A in a least-squares sense. The result of rank lowering is that some dimensions were combined and terms with similar meaning are now connected to each other:
{(car), (truck), (flower)} –> {(1.3452 * car + 0.2828 * truck), (flower)}.

5. This truncation process provides the basis for generating a k-dimensional vector space. Both terms and documents are represented by k-dimensional vectors in this vector space.

6. New documents (e.g., queries) and new terms are represented in the space by a process known as folding-in [16]. When adding a new document, it should be firstly cleaned by the same pre-processing steps (e.g., stopword removal) as those applied to the original documents used in creating the space. The document then is assigned a representation vector that is the weighted average of the representation vectors for the terms of which it is composed. A similar process is applied to fold in new terms.

7. The similarity of any two objects represented in the space is reflected by the proximity of their representation vectors, generally using a cosine measure.

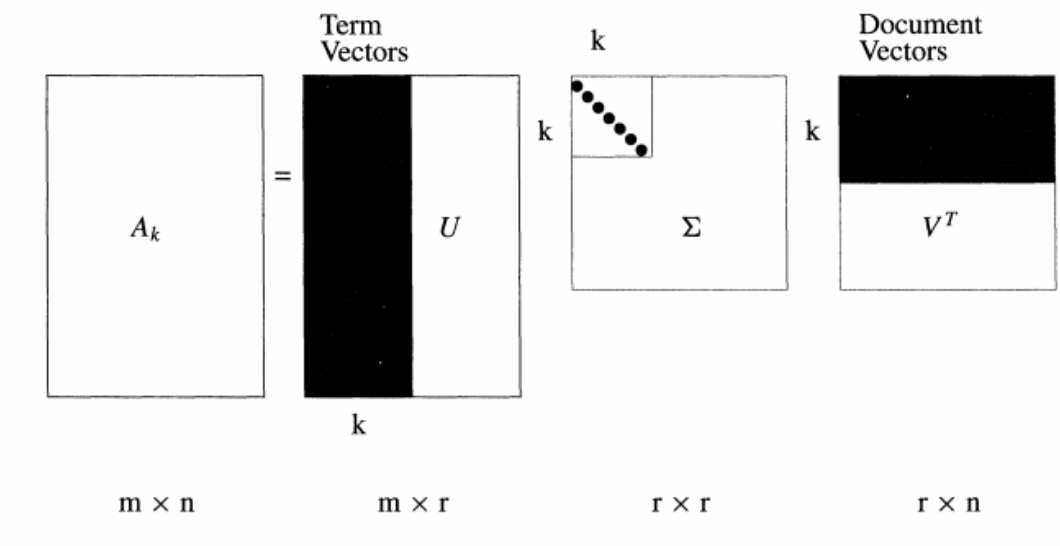


Figure 2.14: Mathematical representation of the matrix A_k. [6]

The low-dimensional vector representation can be further applied to the following challenges:

- Compare the documents in the low-dimensional space (data clustering, document classification).
- Detect relationship between words (synonymy and polysemy)
- Detect similar documents through languages after analyzing a base set of translated documents (cross language retrieval)
- Analyze word relation in document corpus

2.5.3 Word Mover's Distance (WMD)

Word Mover's Distance was introduced in 2015 by [22] as a distance measurement between two sentences or documents. Using pretrained word embeddings [29], it computes the minimum distance which is needed to "transport" the word embeddings from one document to another.

Let us assume we have pretrained word2vec vectors as a matrix $X \in \mathbb{R}^{d \times n}$ for a vocabulary of n words. $x_i \in \mathbb{R}^d$ will be the representation of the i^{th} word in a d-dimensional space. Let the documents be represented as a standard Bag-of-words (nBOW) vectors, $d \in \mathbb{R}^n$ ($d_i = \frac{c_i}{\sum_{j=1}^{n} c_j}$, where c_i is the number of times the word i occurs in the document). Then, let d and d' be the vectors of two semantically closed documents: *"Obama speaks to the media in Illinois"* and *"The President greets the press in Chicago"*. We can assume that these two vectors lie in the $n-1$ dimensional simplex of word distributions and see that the vectors will be placed far from each other because of dissimilarity of the words between them.

The goal of WMD is to embed the semantic similarity between words such as *President* and *Obama* into the document distance metric. It is naturally possible to compute the dissimilarity between words using *Euclidian distance* in the word2vec embedding space. In other words, the distance between words i and j will be $c(i, j) = ||x_i - x_j||$ which can be referred as *cost* for "traveling" from one word to another.

The WMD presents a "transport" matrix $T \in \mathbb{R}^{n \times n}$ in the way that T_{ij} defines how much of word i in d need to be transported to a word j in d'. More precisely, WMD trains T to minimize the cost of moving d to d':

$$\begin{aligned} &\min_{T \geq 0} \sum_{i,j=1}^{n} T_{ij} c(i,j) \\ \text{subject to: } &\sum_{i,j=1}^{n} T_{ij} = d_i \quad \forall i \in \{1, \ldots, n\} \\ &\sum_{i,j=1}^{n} T_{ij} = d_j^{'} \quad \forall j \in \{1, \ldots, n\} \end{aligned}$$

As a result, the documents that share the same or similar words will have smaller distances than documents with uncommon words. In [22] WMD was denoted as a special case of the Earth Mover's Distance metric (EMD) [34] also known as Wasserstein distance [25].

[14] introduced an alleviated version of the transport problem described above by adding an entropy regularizer to the transport objective in order to soothe the cubic time complexity of the Wasserstein distance computation. Given a transport matrix T, let $h(T) = -\sum_{i,j=1}^{n} T_{ij} \log(T_{ij})$ be the entropy of T. For any $\lambda > 0$, the regularized (primal) transport problem is defined as [19]:

$$\begin{aligned} \min_{T \geq 0} & \sum_{i,j=1}^{n} T_{ij} c(i,j) - \frac{1}{\lambda} h(T) \\ \text{subject to:} & \sum_{i,j=1}^{n} T_{ij} = d_i \quad \forall i \in \{1, \ldots, n\} \\ & \sum_{i,j=1}^{n} T_{ij} = d_j^{'} \quad \forall j \in \{1, \ldots, n\} \end{aligned}$$

However, this method does not consider the situation when the number of dimensions is too high. Hence, it will not be efficient for extremely large document settings (all possible words). [22]

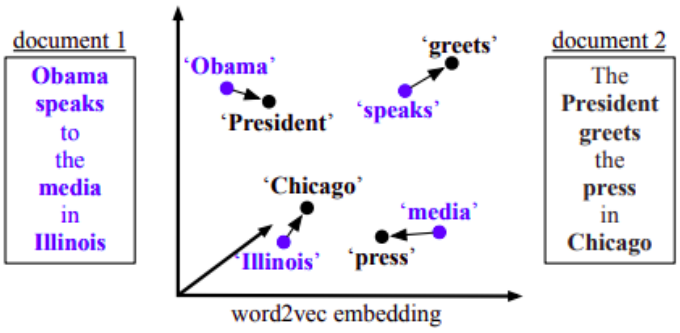


Figure 2.15: An illustration of Word Mover's Distance. [22]

As shown in Figure 2.15, the words in ***bold*** are the embedded word2vec vectors. Firstly, WMD will "transport" each item from document 1 to document 2, since the algorithm is not aware about the linguistic similarity in language yet. Finally, it will select the minimum transportation cost of each word.

The top of Figure 2.16 illustrates the members of the WMD metric between an input sentence D_0 and two sentences D_1, D_2. The arrows in the bottom describe a flow between two words and are labeled with their distance contribution.

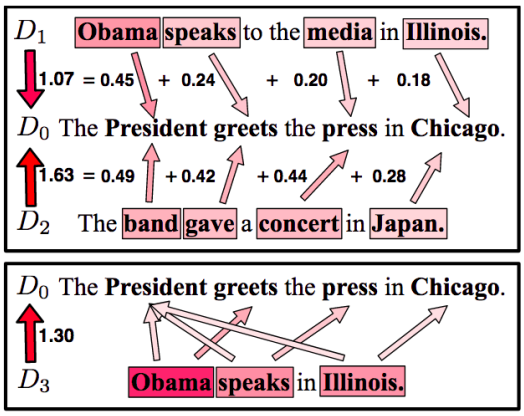


Figure 2.16: WMD computation results. [22]

2.5.4 Clustering - K-means algorithm

Clustering is a popular unsupervised technique of partitioning data points into disjoint subgroups or *clusters*. This is obtained in a way that the points within a cluster are similar to each other and dissimilar to the points in other clusters. [28, 11]

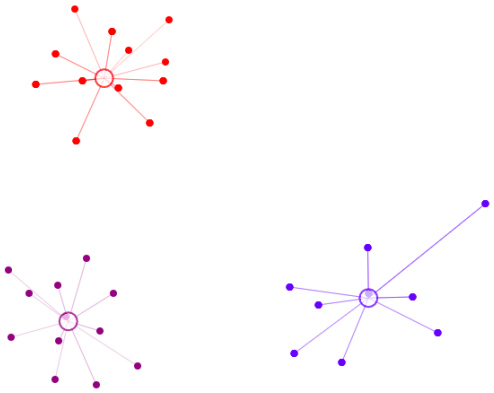


Figure 2.17: An example of clustering.

All clustering methods can be divided into two types:

- **Hard clustering** - each data point can be the member of only one cluster. Example models: *K-means clustering, Hierarchical clustering*
- **Soft clustering** - the probability is assigned for each data point to be in either

of the given number of clusters. Example models: *Expectation-Maximization, Latent Semantic Indexing*

In this section we will cover the most widely used flat clustering technique, namely, *K-means clustering* [20, 26]. Assuming we have a dataset $D = \{x_1, x_2, \ldots, x_N\}$ of N data points, let us define a K for clusters $C = \{C_1, C_2, \ldots, C_k, \ldots, C_K\}$ and c_k will be the centroid of cluster C_k:

$$c_k = \frac{1}{|C_k|} \sum_{x \in C_k} x \tag{2.14}$$

The main objective of K-means algorithm is to divide the data points into K-number of clusters with minimum divergence among points in each cluster called as NP-hard problem [1]. This goal is achieved by minimizing the score of Sum of Squared Errors (SSE):

$$SSE(C) = \sum_{k=1}^{K} \sum_{x_i \in C_k} ||x_i - c_k||^2 \tag{2.15}$$

This function iteratively relocates the centroids and reassigns the data points for clusters until the locally optimal partition set $\hat{C}$ is not reached:

$$\hat{C} = \arg\min_{C} SSE(C) \tag{2.16}$$

K-means algorithm consists of the two following steps:

1. **Initialization:**
   Set c_k for K clusters to random values fetched from the dataset

2. **Iteration (Lloyd's algorithm)**[26]:
   Repeat until convergence criterion is met:

   - **Assignment step**
     Each data point is assigned to the nearest centroid:

     $$P_k^{(t)} = \{x_i : ||x_i - C_k^{(t)}|| \leq ||x_i - C_{k^*}^{(t)}|| \quad \forall k^* = 1, \ldots, K\} \tag{2.17}$$

   - **Update step**
     Calculate the mean of each cluster based on their members and update the centroids:

$$C_k^{(t+1)} = \frac{1}{|P_k^{(t)}|} \sum_{x_i \in P_k^{(t)}} x_i \tag{2.18}$$

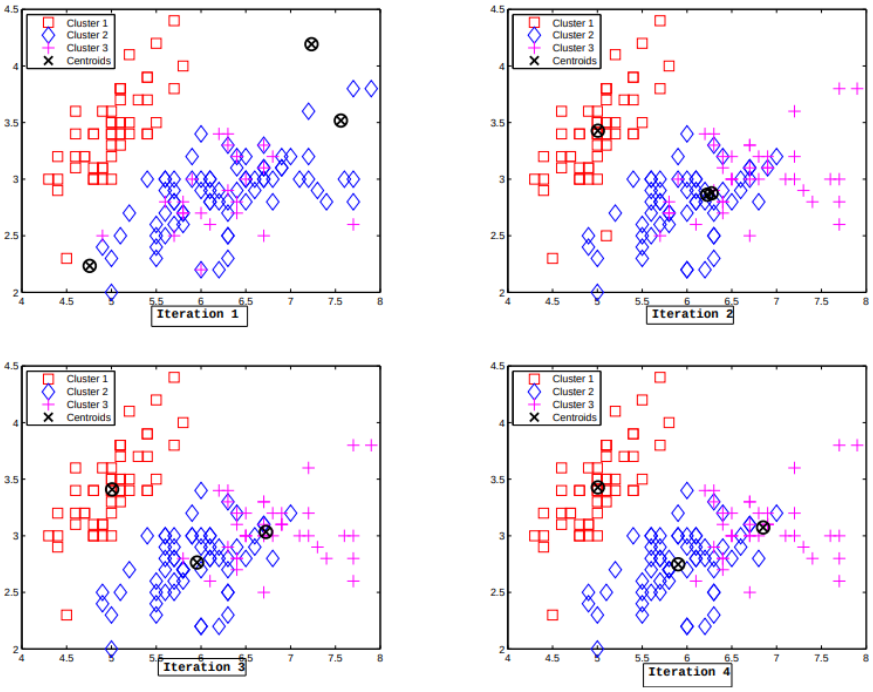


Figure 2.18: An illustration of four iterations of K-means. [31]

The minimizing of SSE proves the convergence of K-means, however, there is no guarantee that a *global minimum* will be reached. This is a particular case when a dataset contains extreme data points that deviate from other values (outliers) and, hence, will not fit well in any cluster. As a result, we end up with a *singleton cluster* which contains only one data point. In order to solve this, we will need to reinitialize the centroids and repeat the steps again.

Chapter 3

Experiments

In this chapter we will explain the data sets used to evaluate the performance of the approaches, the similarity metrics which are used for computing the similarity after the input representation, the proposed approaches used to address the issues introduced in chapter 1 (off-topic detection, plagiarism detection and score propagation) and the experimental results.

3.1 Datasets

One of the major requirements in any data science task is the presence of a rationally chosen data. For our experiments we needed sets of real student essays written in English and fairly scored by professors. After careful review we came up with two datasets:

1. Essays Dataset used in the Automated Student Assessment Prize (ASAP) run by Kaggle [1][2]

2. Corpus of Plagiarised Short Answers created by Paul Clough and Mark Stevenson, University of Sheffield[12]

The dataset 1 will be used for off-topic detection and automatic essay scoring models. The dataset 2 will be used for plagiarism detection model. Both datasets will be discussed thoroughly in the subsections below.

[1] `https://www.kaggle.com/c/asap-aes`

3.1.1 Kaggle dataset

This dataset provides 12976 essays differentiated by eight unique topics . The average essay length ranges from 150 to 550 words per answer. The essays were written by students ranging in grade levels from Grade 7 to Grade 10. All works were scored by real teachers and were double-scored. Each of the eight essay sets has its own unique properties and was constructed using a corresponding prompt, each with individual marking criteria and score range.

The training set of all essays consists of the following columns:

- **essay_id:** A unique identifier for each individual student essay
- **essay_set:** 1-8, an id for each set of essays
- **essay:** The ascii text of a student's response
- **rater1_domain1:** Rater 1's domain 1 score; all essays have this
- **rater2_domain1:** Rater 2's domain 1 score; all essays have this
- **rater3_domain1:** Rater 3's domain 1 score; only some essays in set 8 have this.
- **domain1_score:** Resolved score between the raters; all essays have this
- **rater1_domain2:** Rater 1's domain 2 score; only essays in set 2 have this
- **rater2_domain2:** Rater 2's domain 2 score; only essays in set 2 have this
- **domain2_score:** Resolved score between the raters; only essays in set 2 have this
- **rater1_trait1 score - rater3_trait6 score:** trait scores for sets 7-8

3.1.2 Clough & Stevenson dataset

The corpus was developed with a purpose to address plagiarism as an increasing problem for higher education institutions. It consists of short answers to Computer Science prompts where plagiarism can be detected. The dataset is constructed in the way of representing several stages of plagiarism.

Prompt	#Essays	Avg length	Scores
1	1,783	350	2-12
2	1,800	350	1-6
3	1,726	150	0-3
4	1,726	150	0-3
5	1,805	150	0-4
6	1,800	150	0-4
7	1,569	250	0-30
8	723	650	0-60

Table 3.1: Statistics of the Kaggle dataset [38]

This corpus can be successfully used in plagiarism detection systems for the reason that it simulates typical cases of plagiarism practised by students as realistically as possible. There is a set of five prompts (A-E) on five different topics that can be possibly included in the Computer Science curriculum. For each of these prompt a set of responses were provided using various simulating approaches. Based on these approaches the one can denote whether the answer is plagiarised or not. To simulate plagiarism authors used a set of texts from Wikipedia which participant will use as a source to produce a plagiarised answer.

Four stages of plagiarism are described as follows:

- **Near copy:** Participants copied text from the corresponding Wikipedia article (i.e. copy-paste). No guidance was provided about which parts of the article to copy. As a result, short text with a length of 200-300 words had to be produced by multiple selections from the article.

- **Light revision:** Participants based their responses on the texts from Wikipedia article without instruction on which parts to select. However, they were asked to do paraphrasing by using synonyms and changing grammatical structure in some phrases. The order of copied information had to be preserved.

- **Heavy revision:** Participants based their responses on the corresponding Wikipedia article but were asked to rephrase the entire text with synonyms and change the structure. They were also allowed to partition a source sentence or combine various source sentences all together in an arbitrary manner.

- **Non-plagiarism:** Participants had to prepare for the questions with the help of relevant learning materials. They further produced their unique answers based on the information they had learned from the materials. They were also allowed to look at some additional materials during the exam but were asked to completely ignore the Wikipedia articles.

The authors invited 19 participants to take part in creating the corpus. All of them were students in Computer Science Department of Sheffield University since each participant had to have at least basic understanding of Computer Science. The answers had to be in a range from 200 to 300 words containing only standard (ASCII) characters. Each participant had to answer five questions with different approaches: two of the five questions were answered without plagiarising (the "non-plagiarism" category), one question using the near copy, one using light revision and one using heavy revision.

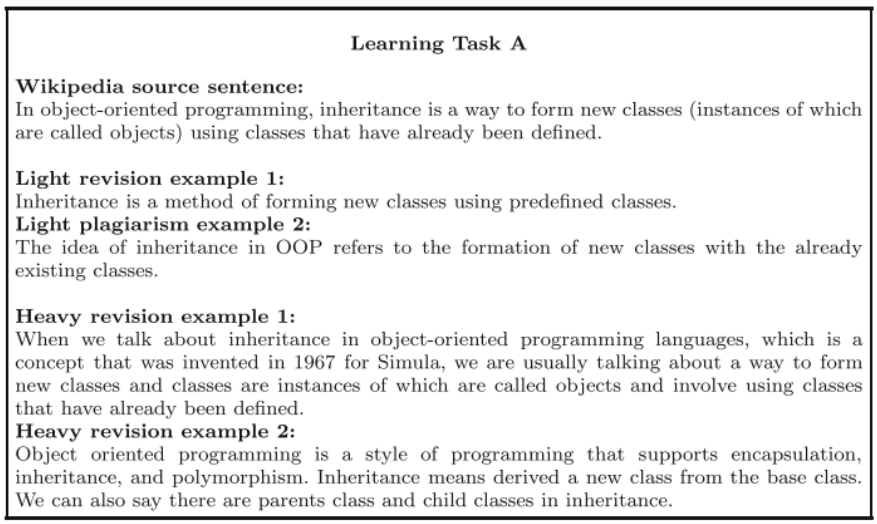
Category	Learning task					Total
	A	B	C	D	E	
Near Copy	4	3	3	4	5	19
Light revision	3	3	4	5	3	19
Heavy revision	3	4	5	4	3	19
Non-plagiarised	9	9	7	6	7	38
Total	19	19	19	19	19	95

Table 3.2: Number of answers by learning task and plagiarism category [12]

The corpus consists 95 answers (Table 3.2) and five Wikipedia articles which is 100 documents in total. The values for the categories in the dataset are the following: "cut" - near copy, "light" - light revision, "heavy" - heavy revision, "non" - non-plagiarism.

3.2 Similarity metrics

This section will describe the similarity metrics we used in the experiments. In order to find the similarity between two document vectors, we need a similarity measure



Learning Task A

Wikipedia source sentence:
In object-oriented programming, inheritance is a way to form new classes (instances of which are called objects) using classes that have already been defined.

Light revision example 1:
Inheritance is a method of forming new classes using predefined classes.
Light plagiarism example 2:
The idea of inheritance in OOP refers to the formation of new classes with the already existing classes.

Heavy revision example 1:
When we talk about inheritance in object-oriented programming languages, which is a concept that was invented in 1967 for Simula, we are usually talking about a way to form new classes and classes are instances of which are called objects and involve using classes that have already been defined.
Heavy revision example 2:
Object oriented programming is a style of programming that supports encapsulation, inheritance, and polymorphism. Inheritance means derived a new class from the base class. We can also say there are parents class and child classes in inheritance.

Figure 3.1: An example of light and heavily revised answers to the task A. [12]

which will calculate the distance between them. The lower the distance, the higher is the similarity.

3.2.1 Cosine similarity

Cosine similarity is one the most widely used approaches to compute the similarity between documents. It uses the cosine of angle between two vectors particularly used in positive space where the result is between 0 and 1.

In our experiments we used cosine similarity metric to calculate the distance between the topic words extracted from LDA model and student essays which were firstly converted to Tf-idf matrix and then fed into the LSA model. Given two LSA vectors a and b, the cosine similarity is the following:

$$Cos(a, b) = \frac{\vec{a} \cdot \vec{b}}{\|\vec{a}\| \|\vec{b}\|} \tag{3.1}$$

As shown in the Figure 3.2, the documents "Python is cool" and "I like football" head up to the opposite directions, hence, the cosine angle between them will be large and they are considered as dissimilar. It is also important to note that according to the LSA vector representation, the document "Python is cool" will be very similar to other documents written about machine learning as well as the document "I like football" will be similar to the documents about sport. This can be explained by

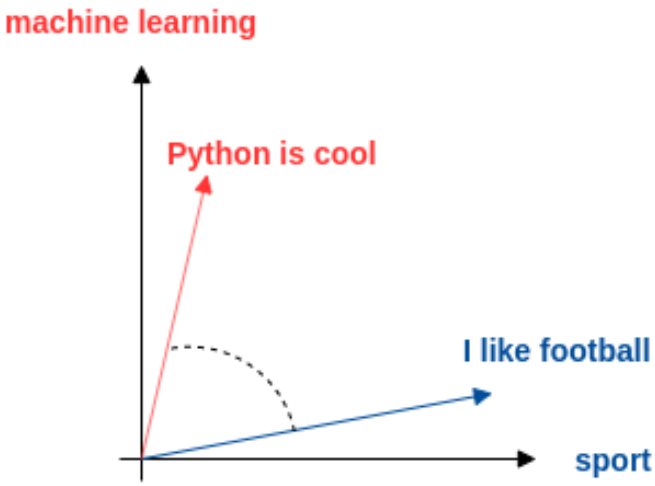


Figure 3.2: An illustration of Cosine similarity

the LSA power to combine different words occurring in the same contexts.

3.2.2 Earth Mover's Distance

Earth Mover's Distance metric is a default dissimilarity measure in Word Mover's Distance algorithm [40] (discussed in 2.5.3). It computes the dissimilarity between two multi-dimensional distributions in some vector space (e.g. cloud of word embeddings) where the distance between points is measured by, so called, ground distance (e.g. Euclidean distance).

The EMD can be considered as a solution to the well-known transportation problem. Based on our experiment in WMD model, we assumed the distributions as the word distributions of documents, the weighted graph represents the similarity between two documents and the EMD is trained to calculate the minimum cumulative cost to "transport" all words from one document to another.

Given two documents A and B let us define a weighted graph G as follows [44]:

- Let $A = \{(t_{a1}, w_{a1}), (t_{a2}, w_{a2}), \ldots, (t_{am}, w_{am})\}$ be the representation of the document A; t_{ai} is a unique word in the document A and w_{ai} is the word's weight over the term-document matrix.

- Let $B = \{(t_{b1}, w_{b1}), (t_{b2}, w_{b2}), \ldots, (t_{bn}, w_{bn})\}$ be the representation of the document B; t_{bj} is a unique word in the document B and w_{bj} is the word's weight over the term-document matrix.
- Let $D = \{d_{ij}\}$ be the distance matrix where d_{ij} is the semantic distance between words t_{ai} and t_{bj} computed beforehand using Euclidean distance measure on pretrained word2vec embeddings.
- Let $G = \{A, B, D\}$ be the weighted graph produced by the combination of A, B and D.

Based on the weighted graph G, we can define a flow $F = \{f_{ij}\}$ where f_{ij} is the flow between the words t_{ai} and t_{bj} that minimizes the overall transportation cost:

$$WORK(A, B, F) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij} \tag{3.2}$$

After having the transportation problem solved and finding the optimal F, the EMD can be defined as:

$$EMD(A, B) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} \tag{3.3}$$

The last step is to compute the similarity between documents A and B:

$$Sim_{EMD}(A, B) = 1 - EMD(A, B) \tag{3.4}$$

The normalization of $Sim_{EMD}(A, B)$ is in the range of [0,1] such that the higher $Sim_{EMD}(A, B)$ is, the more the documents A an B have in common.

3.3 Performance Measures

In order to analyse and evaluate the performance of machine learning algorithms, we need some performance metrics. Our challenge is considered as classification problem, hence, we will be using such metrics as confusion matrix, accuracy, precision and recall. Before diving into the description of each metric, let us define the basic terminology of the evaluation techniques such as *true positives*,*true negatives*,*false*

positives and *false negatives*. They demonstrate the correlation between the results of the classifier and actual observations. The terms *positive* and *negative* refer to the prediction made by the classifier, whereas *true* and *false* refer to whether that prediction matches to the actual observation.

- **True Positives (TP) -** when both the actual and predicted values are positive.
- **True Negatives (TN) -** when both the actual and predicted values are negative.
- **False Positives (FP) -** when the actual value is negative and the predicted value is positive.
- **False Negative (FN) -** when the actual value is positive and the predicted value is negative.

1. **Confusion matrix**
   Confusion matrix, also called as an error matrix [37], is a table layout that visualise the performance of a machine learning algorithm. The actual class is represented by rows and the predicted class performs as columns. Here is an illustration of confusion matrix:

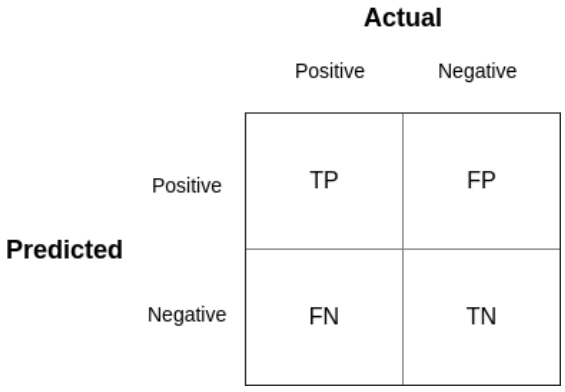


Figure 3.3: Confusion matrix

2. **Accuracy**
   Accuracy computes the coefficient of match between the actual and predicted classes [41]. It is a proportion of all the true results over the total number of predictions.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{3.5}$$

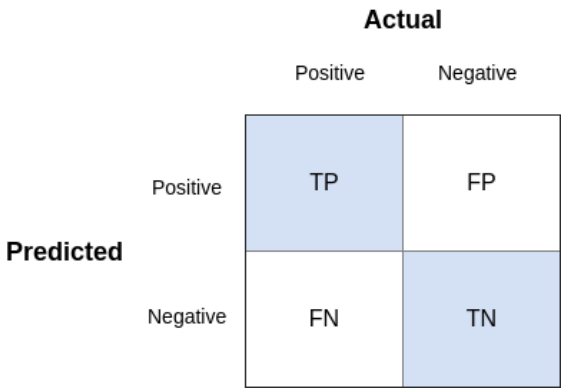


Figure 3.4: Accuracy

3. **Precision**

   Precision describes how precise or accurate the model is by calculating the ratio of True Positives over all the positive predictions [41]. The usage of Precision metric is relevant when the cost of False Positives is high.

$$Precision = \frac{TP}{TP + FP} \tag{3.6}$$

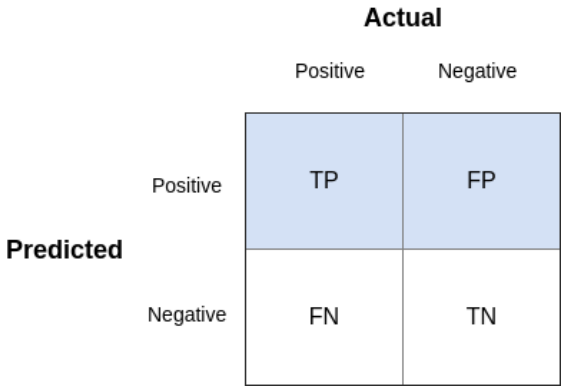


Figure 3.5: Precision

4. **Recall**

   Recall is a metric which computes how many of the True Positives the model managed to capture over the total number of actual positive observations. This metric is quite useful to apply when there is a high cost of False Negatives.

$$Recall = \frac{TP}{TP + FN} \tag{3.7}$$

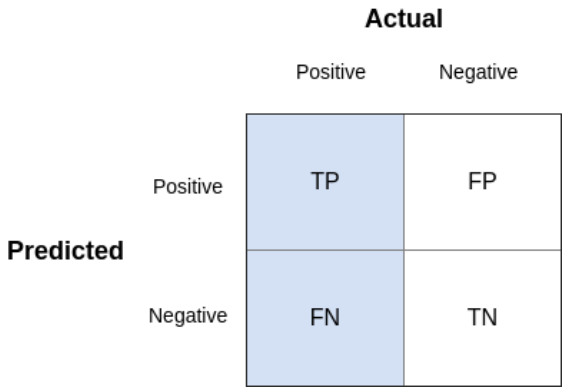


Figure 3.6: Recall

3.4 Implementation details

3.4.1 Packages used

This section will provide the detailed review about the main implementation tools we used in our proposed models. The models and data preprocessing were run by Python 3.7.3. Over the recent years, Python has become one of the popular programming languages and the top choice language for developers in machine learning, artificial intelligence (AI) and deep learning projects. The main aspect that has made Python such a well-known solution, is its variety of useful libraries and frameworks that helps developers easily implement machine learning tasks and save time of coding. In our implementations we used the popular packages provided by Python community such as NumPy, SciPy, Pandas, Gensim [2], Scikit-learn [3] and NLTK [4].

NumPy is the most fundamental module provided for scientific computations in Python. It consists of a variety of essential features for solving multidimensional

[2]https://radimrehurek.com/gensim/
[3]https://scikit-learn.org/stable/
[4]http://www.nltk.org

array and matrix problems and also enables efficient implementation of mathematical computations in a high-level language. The package has been widely adopted in academia, national laboratories, and industry, with applications ranging from gaming to space exploration [43]. In our case the NumPy was used for representing the actual and predicted data as arrays after fetching the results from model.

SciPy is a widely used library for ML projects which contains substantial modules for linear algebra, optimization, integration, statistics and image processing. The functionality of SciPy library is based on NumPy such that its arrays make extensive use of NumPy. Thus, the data structure of SciPy is a multidimensional array provided by NumPy. In our LSA model we used a submodule of SciPy to compute cosine similarity between documents in LSA matrix.

Pandas was designed as an exclusive solution to work with data in a simple and intuitive way. It contains an abundance of functions used for data manipulation, aggregation and visualization. Pandas library can directly operate on data fetched from different data sources such as SQL databases, CSV, Excel. Pandas functions play an essential role in all the text mining projects as it handles the crucial operations of preprocessing task having the data ready for further implementations. There are two main structures in Pandas: "Series" and "DataFrame". Series is a one dimenstional labeled array like object, while DataFrame is a 2-dimensional labeled data structure.

Gensim was primarily presented as a Natural Language Processing package that deals with "Topic Modeling for Humans" [32]. Thus, Gensim provides the implementation of the most popular topic modeling techniques as LDA and LSA. Moreover, Gensim is considered as an ideal solution for tasks such as converting words to word vectors (Word2Vec, FastText), building paragraph vectors (Doc2Vec), finding text similarity and text summarization. It also has implementation of non-negative matrix factorization (NMF), tf-idf and random projections. The most substantial advantage of Gensim is that it can easily handle large text collections without having to load the entire file in memory which differentiates it from most other ML libraries that work only with in-memory processing. In our experiments we used Gensim for implementing LDA, LSA models, loading the pretrained Word2Vec vectors and train Doc2Vec on text documents.

Scikit-learn is a popular Python library that supports supervised and unsuper-

vised learning tasks. The basic functionality of Scikit-learn was derived from previously discussed packages NumPy and Scipy, hence, it is fully compatible with them. Many useful adds-on have been designed for common ML algorithms such as clustering, regression and classification including support vector machines, random forests, gradient boosting, k-means and DBSCAN. We used Scikit-learn to measure the distance between documents, construct term-document matrix, provide performance measure report (e.g. confusion matrix, accuracy) and for clustering.

NLTK (stands for Natural Language Toolkit) is one of the most powerful libraries in Python for statistical NLP to operate on English written texts [7]. It provides packages to make machines understand human language and give a relevant response to it. This is achieved by such techniques as tokenization, stemming, lemmatization, punctuation, tagging, parsing, character count and word count supported in NLTK. This library was successfully used in our experiments for text pre-processing tasks.

3.5 Off-topic essays detection

Every user input has to pass through a proper validation mechanism before computing the score to increase the use-ability of AEE systems. Every well-written essay that does not address the question topic may receive a good score from an AEE system because of linguistic features, such as text structure and surface [18]. One of the issues to be addressed is the detection of essay written out of the context of the prompt, called off-topic. Off-topic essay detection is an identification of student essay which is not written to the test question topic. It has great significance to get the user's confidence in AEE systems and also for improving the fairness, robustness, and accuracy of AEE systems. Such problems should be addressed by using off-topic essay detection methods alongside with AEE systems[18] to build the trust of the user's of AEE systems. In our work, we proposed and implemented two-hybrid approaches by combining Latent Dirichlet Allocation with Latent Semantic Analysis and Word Movers Distance namely LDA_LSA and LDA_WMD for off-topic detection. In both of the methods, LDA was used to extract topic words for each prompt.

To evaluate the performance of the proposed approaches, the kaggle dataset (discussed in 3.1.1) of student essays was used in the implementation. Firstly, we generate a reference answer for a target essay set as the dataset we used did not

have actual reference answers using Latent Dirichlet Allocation. From the kaggle dataset, we chose three sets of essays (sets 4, 5 and 6) written on a specific topic which we think are appropriate for this task. The off-topic essays in the data set mainly consist of two sources. One is selected from the original data set and was drawn from among a set of essays assigned a score of 0 by human graders, and the second one is by randomly selecting essays from other topics to extend the number of off-topics for each set.

```
Topic: 0
Words: 0.048*"build" + 0.041*"dirig" + 0.027*"would" + 0.026*"state" + 0.023*"obstacl" + 0.022*"empir" + 0.019*"ma
st" + 0.018*"moor" + 0.017*"dock" + 0.016*"builder" + 0.015*"face" + 0.014*"wind" + 0.012*"could" + 0.011*"area" +
0.010*"top" + 0.010*"hydrogen" + 0.010*"law" + 0.009*"use" + 0.009*"mani" + 0.009*"allow" + 0.009*"problem" + 0.00
8*"new" + 0.008*"'s" + 0.007*"attempt" + 0.007*"fli" + 0.006*"low" + 0.006*"airship" + 0.006*"urban" + 0.006*"air"
+ 0.006*"one" + 0.006*"anoth" + 0.006*"around" + 0.006*"york" + 0.006*"frame" + 0.006*"flammabl" + 0.005*"natur" +
0.005*"current" + 0.005*"safeti" + 0.005*"violent" + 0.005*"also" + 0.005*"highli" + 0.005*"stress" + 0.005*"make"
+ 0.005*"even" + 0.005*"due" + 0.005*"blimp" + 0.004*"shift" + 0.004*"ship" + 0.004*"weight" + 0.004*"helium" + 0.
004*"made" + 0.004*"back" + 0.004*"idea" + 0.004*"constantli" + 0.004*"land" + 0.004*"popul" + 0.004*"citi" + 0.00
4*"danger" + 0.004*"reason" + 0.003*"caus" + 0.003*"fire" + 0.003*"safe" + 0.003*"n't" + 0.003*"architect" + 0.003
*"paragraph" + 0.003*"illeg" + 0.003*"way" + 0.003*"tie" + 0.003*"excerpt" + 0.003*"tether" + 0.003*"exist" + 0.00
3*"outsid" + 0.003*"first" + 0.003*"high" + 0.003*"pedestrian" + 0.003*"swivel" + 0.003*"add" + 0.003*"realiz" +
0.003*"thousand" + 0.003*"rather" + 0.003*"peopl" + 0.002*"much" + 0.002*"hindenburg" + 0.002*"dens" + 0.002*"plac
e" + 0.002*"greatest" + 0.002*"accid" + 0.002*"lead" + 0.002*"practic" + 0.002*"construct" + 0.002*"never" + 0.002
*"unit" + 0.002*"issu" + 0.002*"get" + 0.002*"tri" + 0.002*"modifi" + 0.002*"..." + 0.002*"pressur" + 0.002*"like"
+ 0.002*"strengthen"
```

Figure 3.7: Topic words extracted from the best-scored essays

The words shown in Figure 3.7 were fetched from the best essays of set 6. The raw data was firstly preprocessed by tokenizing, stemming and removing stopwords and fed into LDA in a shape of Bag-of-words model for each document. These words will be later play a role of a reference answer for the original essay set.

The target set does not have enough off-topics, hence, we decided to populate the off-topic data by adding the essays from the other sets.

Original Prompt	**Other Prompts**	**On-topic essays**	**Off-topic essays**
The Obstacles the Builders of the Empire State Building faced in attempting to allow dirigibles to Dock	Winter Hibiscus by Minfong Ho (first case)	1756	1816
	The Blueprints of Our Lives by Narciso Rodriguez (second case)	1756	1849

Table 3.3: Our Datasets.

The original essay set contains 1800 documents in total having 1756 on-topic essays and 44 off-topic essays. Other prompts represent two sets of essays written

on different topics that should be treated as off-topic essays when comparing to the reference answer which was extracted from the original essay set.

As was already mentioned above, we processed the off-topic detection in two models: LSA and WMD. The data for LSA was represented as Tf-idf matrix of the preprocessed documents. Finally, the LSA vector of a reference answer was compared to all the LSA vectors including the original essay set and the essay set written in another topic. In case of WMD model, the pretrained word embeddings were used containing 3 millions of word vectors from Google News. Having the word vectors for each of the document, we computed the similarity between our reference answer and all the documents from the original topic and another topic. [39]

3.5.1 Results and discussion

This section describes the results of the experiment carried out to detect off-topic essays. As you can see from the Tables 3.4 and 3.5, both LSA and WMD models performed well in detecting off-topic essays. However, independently of the content of off-topics, LSA model performed with 98% accuracy in both cases, whereas WMD performed with 95% and 91% accuracy. The precision of LSA is estimated as 96% and 97% which means that in both cases the rate of False Positives was low. For WMD the precision is 97% in the first case which is quite good. However, it has only 90% of precision in the second case which indicates high rate of False Positives. Recall for LSA comprises 99% and 100% in the first and second cases, while in WMD the recall is 92% in both cases. It means that LSA has a lower rate of False Negatives compared to WMD. It is recommended to have a low rate of False Negatives in Automatic Essay Evaluation system as we should be more concerned about misleading a student that he/she has written an off-topic essay rather than about missing an off-topic essay [18].

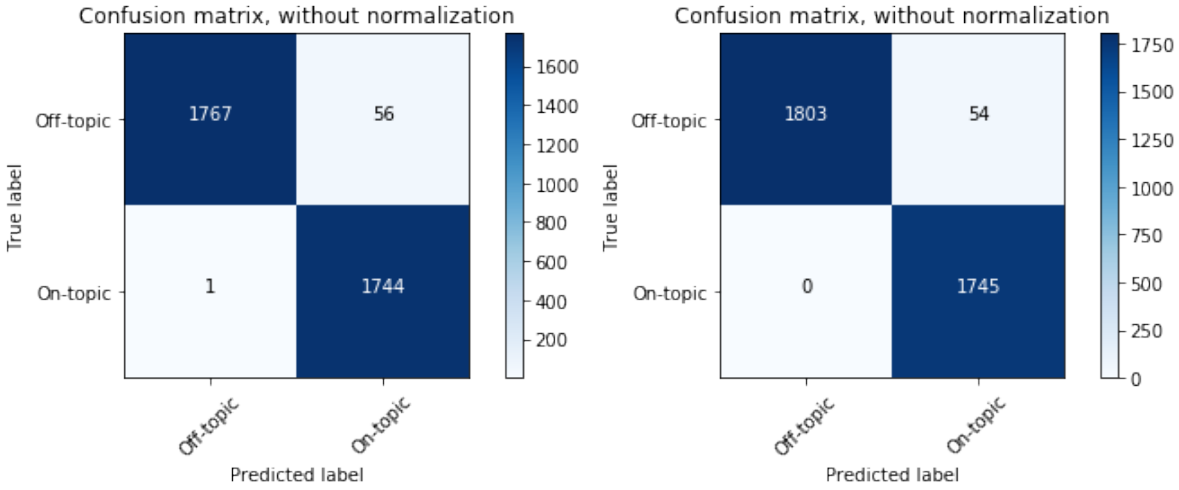
Performance metrics	**LDA+LSA**	**LDA+WMD**
Accuracy	0.984	0.951
Precision	0.968	0.973
Recall	0.999	0.924

Table 3.4: Performance statistics of the first case.

Performance metrics	LDA+LSA	LDA+WMD
Accuracy	0.985	0.916
Precision	0.970	0.906
Recall	1.000	0.924

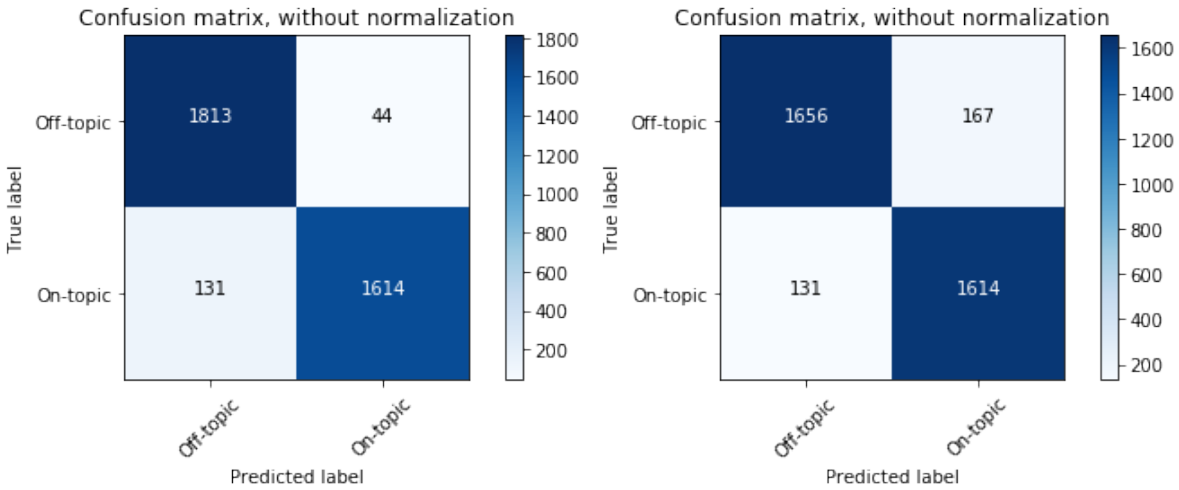
Table 3.5: Performance statistics of the second case.

The figures 3.8 and 3.9 represent the non-normalized confusion matrices of the actual and predicted documents.



(a) *first case* **(b)** *second case*

Figure 3.8: Confusion matrix of the LDA+LSA results



(a) *first case* **(b)** *second case*

Figure 3.9: Confusion matrix of the LDA+WMD results

The main objective of this experiment was to check the performance of widely accepted models in semantic text similarity on off-topic essay detection without need of a specific large amount of training data. The results showed that both models distinguished one topic from another very well. Nonetheless, with very similar rate of False Positives in both models, the LSA model excelled having relatively low rate of False Negatives as opposed to WMD model.

3.6 Plagiarism detection

Plagiarism is one of the growing issues in academic and research field, raising more concerns in evaluating students performance in teaching and learning using essay exams . During submitting solutions to essay exams , students may commit plagiarism by copying and pasting solutions from other students. It has become a very common issue when evaluating the students work and their creativeness. To address this issue, One approach using lexical similarity(Tf-idf) and two other approaches using semantic similarity(LSA and WMD). The experimental results show that lexical similarity approach using tf-idf and cosine similarity performed well than the other semantic similarity approaches. The Clough & Stevenson dataset 3.1.2 was used in this experiment. As shown in the Table 3.1, there were 19 answers per-

formed for each of 5 tasks. There were also the reference answers for each task taken from Wikipedia articles so we could predict the level of plagiarism comparing the reference answer to the answers of participants. Thus, in all the three methods we implemented (WMD, LSA, Tf-Idf), the similarity between the reference document and the simulated document was computed and based on the appropriate threshold, a certain level of plagiarism was designated to the given simulated document.

3.6.1 Results and Discussion

Task	**WMD**	**LSA**	**Tf-Idf+Cosine**
A	0.68	0,73	0.84
B	0.31	0.63	0.68
C	0.36	0.52	0.58
D	0.57	0.63	0.68
E	0.52	0.63	0.74

Table 3.6: Accuracy Performance of the implemented approaches

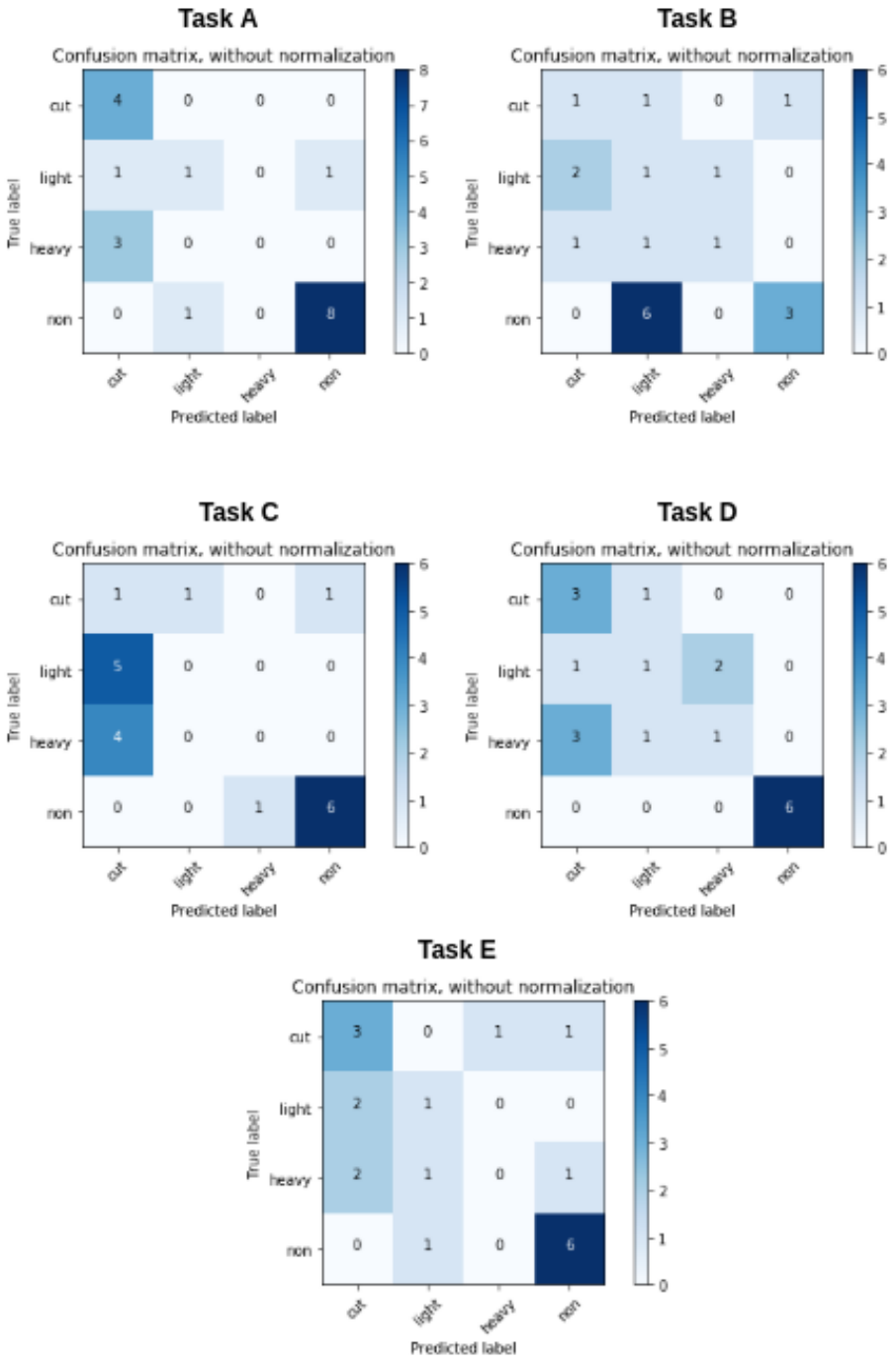


Figure 3.10: Confusion matrix of the WMD results

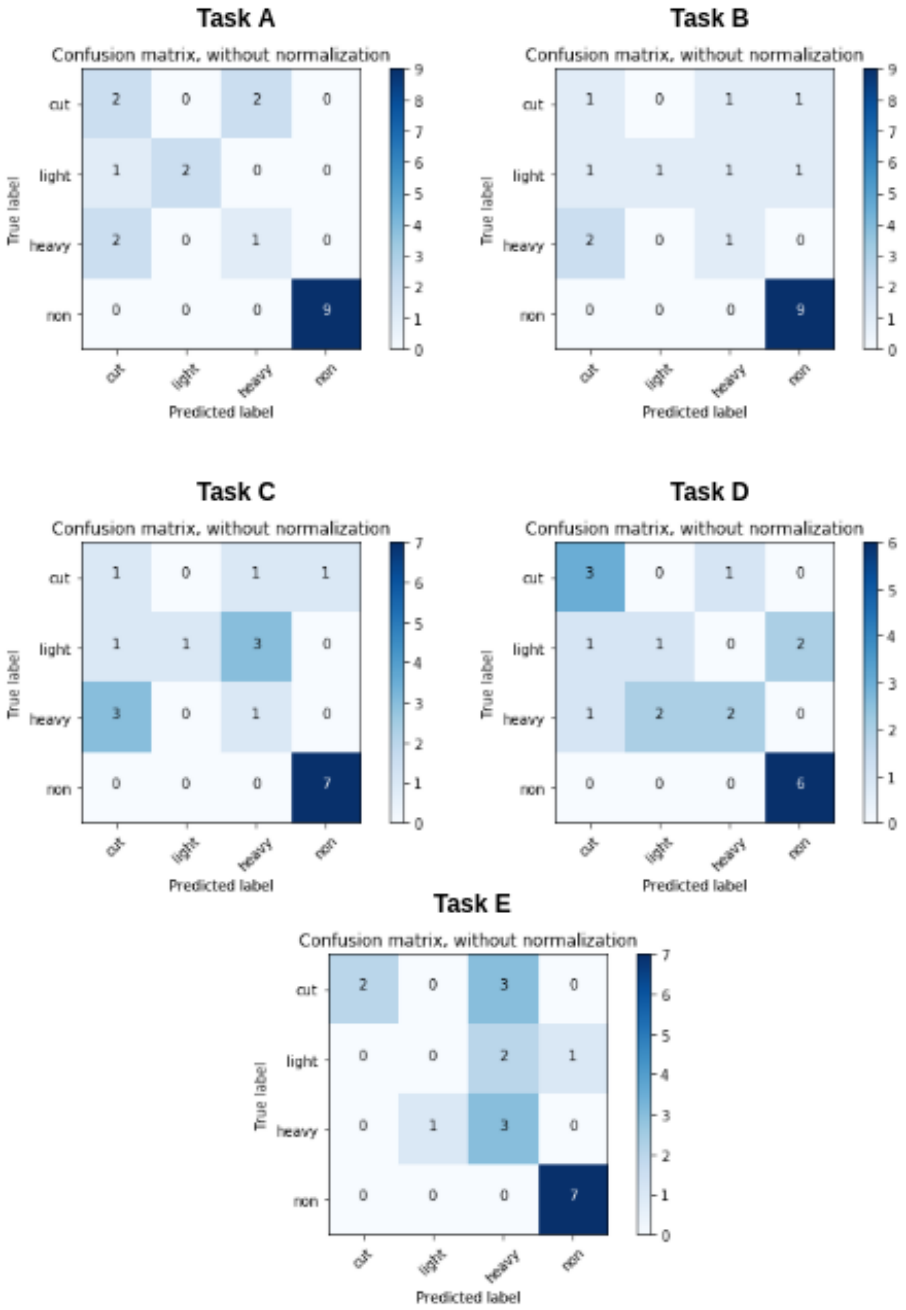


Figure 3.11: Confusion matrix of the LSA results

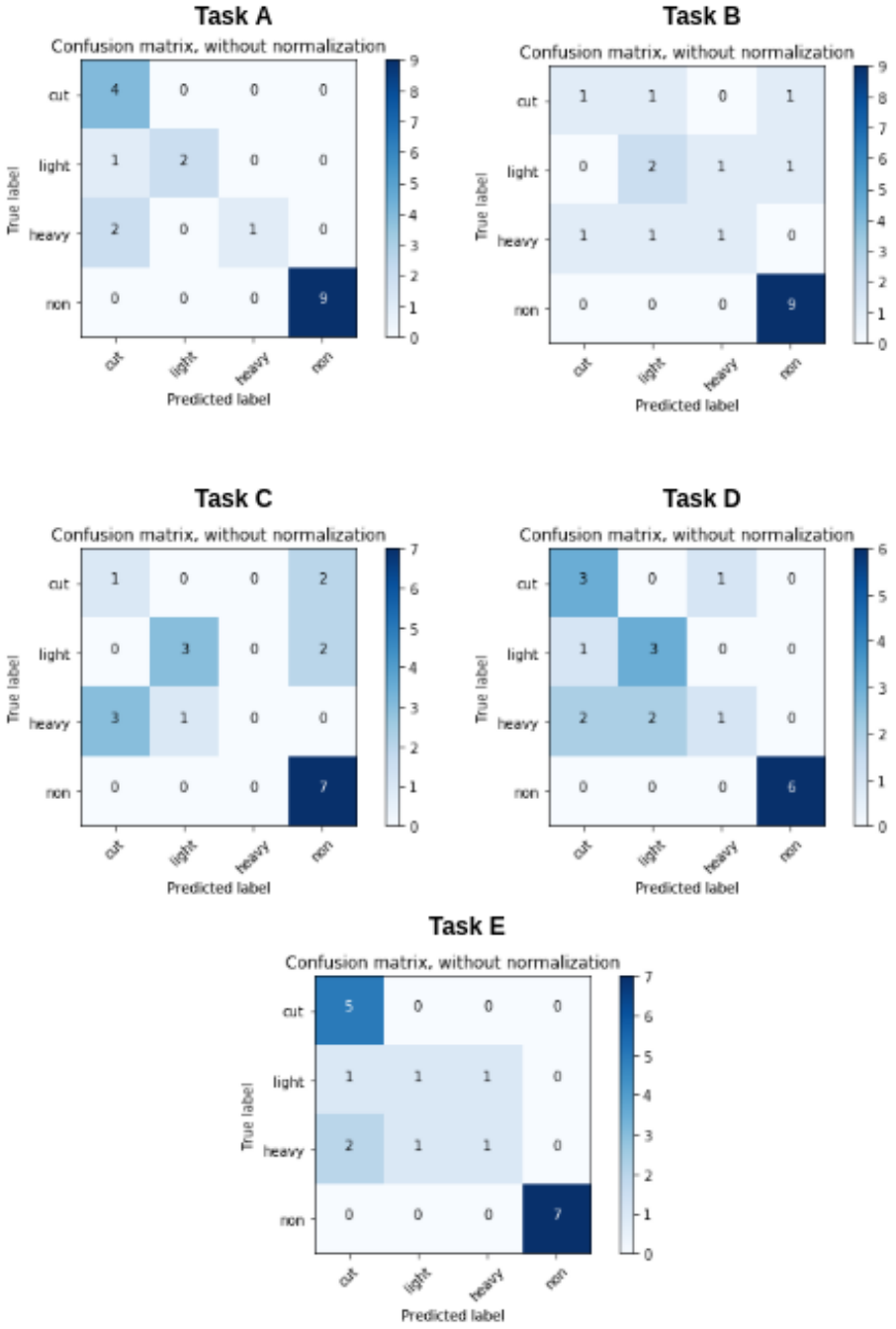


Figure 3.12: Confusion matrix of the Tf-idf+Cosine results

As shown in the Table 3.6, a simple Tf-Idf combined with Cosine similarity performed better than WMD and LSA. The problem with semantic models in this implementation is that they tend to have a quite high similarity result between texts which are written on the same topic but have different lexical and grammatical structures. In other words, the similarity difference between the five categories of plagiarism was not too visible and in some cases it overlapped. For example, the non-plagiarised answer might be more similar to the prompt than a lightly plagiarised answer which is actually wrong. This situation is illustrated in the Figure 3.10 in the matrix for Task B where six non-plagiarized answers have been predicted as lightly plagiarised. When it comes to Tf-idf, it can easily distinguish the non-plagiarised answers from others because it is not focusing on the context but only bases its measure on the frequency of words in the documents. The distinction between other categories was also good. In the Figure 3.12 in Task D three "cut" and three "light" answers were predicted correctly.

The objective of this task was to check which method of document similarity would be more suitable for plagiarism detection. As results showed, Tf-idf combined with cosine similarity performs better than semantically focused models.

3.7 Score propagation

The objective this section is to find a simplest and easiest way to score essays with small amount of labeled essay sets. To achieve the objective, we used clustering methods to automatically cluster essay into different clusters and the teacher will only score the centriods' of each cluster and the score of the centriod will then be propagated to other members of the cluster.

Therefore, we implemented K-means clustering method to partition the set of documents in order to further predict the scores for each cluster. The kaggle dataset was used in this experiment. The first and second essay sets were chosen which contain the human scores with a range of $[2 - 12]$ and $[1 - 6]$ respectively. The following steps were performed during this experiment:

1. We used three different approaches, i.e. Latent semantic Analysis(LSA), Word2vec and Doc2vec, to semantically represent the input essay set.

2. After learning the feature vectors semantically, we used K-means clustering to cluster and further to sub cluster the essay.

3. Then, the true score of each centroid was propagated to the other members of each cluster.

3.7.1 Results and discussion

The experimental results of the proposed approaches are presented in Table 3.7, Figure 3.13 , Figure 3.14 and figure 3.15.

LSA	Word2Vec Mean	Doc2Vec
0.47	0.48	0.54
0.49	0.54	0.55

Table 3.7: Accuracy performance of the implemented approaches

The accuracy performance in the Table 3.7 shows that all the three proposed approaches had similar results. However, Doc2Vec outperformed in both cases having 54% and 55% of accuracy respectively.

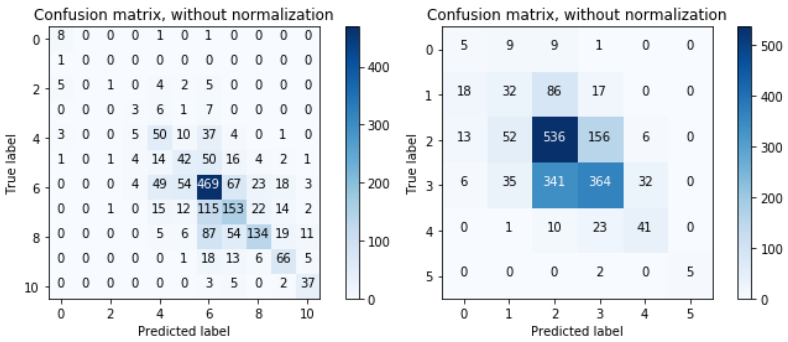


Figure 3.13: Confusion matrix of the Doc2Vec results

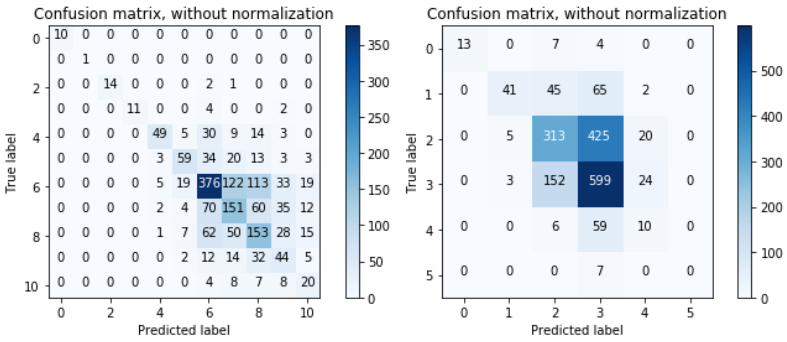


Figure 3.14: Confusion matrix of the Word2Vec Mean results

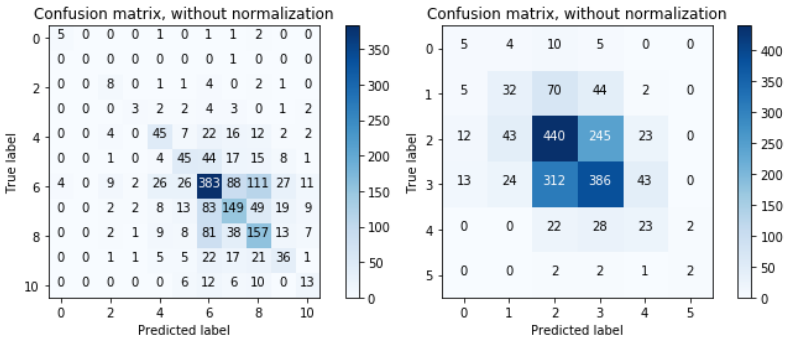


Figure 3.15: Confusion matrix of the LSA results

The overall result showed that around half of scores have been predicted by clustering the documents. Apparently, the result is not brilliant. However, the performance in Figures 3.13, 3.14 and 3.15 shows that the highest errors are concentrated among the grades that are closed to each other. For example, in the first case of all the three confusion matrices it is shown that the highest errors are concentrated in confusing the scores between 6, 7 and 8. Meanwhile, the second case illustrates the highest concentration of the errors in confusing between the scores 2 and 3. This can be explained by a small difference between the essays that have close scores, hence, these kind of essays might be designated to the wrong clusters. Still, most existing supervised-based evaluation approaches utilize around 70% of training data to achieve a reasonable performance results of $80 - 90\%$ accuracy. In our case, both sets were firstly partitioned by five clusters that in their turn were partitioned by 50 clusters each. Thus, only 250 points out of around 1800 points of data (15%) were utilized to get the actual scores for further propagating them to the rest of essays. We have achieved around 50% of accuracy which is actually quite good having that

much of data.

Chapter 4

Conclusion and Future Work

This study aimed to enhance the use-ability of Automatic Essay Evaluation (AEE) systems by implementing unsupervised prediction algorithms without having a specifically labeled training data and also by integrating features for validating inputs to the AEE system. These algorithms will help a user of AEE to filter out "bad essays" (off-topic or plagiarised) and also minimize the effort of manual annotation of a large amount of student essay solutions.

In the experiment for detecting off-topic essays, we implemented two hybrid models LDA_LSA and LDA_WMD. Firstly, we generated a reference answer using LDA from the essay set that we considered as original and written on the topic of the essay question. Then, we constructed an off-topic dataset having the off-topic essays from the original set which have a score of zero and a randomly chosen set written on a different topic. WMD and LSA were used as a model for identifying either the essay is on-topic or off-topic using the similarity threshold value determined during the experiment. The results showed that both models performed similarly well, filtering out the essays that were not written on the generated reference answer with fairly low rates of False Positives and False Negatives. However, the LDA_LSA model outperformed having a lower rate of False Negatives.

To detect student essays which are "plagiarised", we proposed and implemented two context level models (WMD and LSA) and one with Lexical model (simple TF-IDF approach). according to our experimental results, a simple lexical approach using TF-IDF representation performed well than the other models which consider the semantic or contextual similarity between the essays.

The last experiment was aimed at building the algorithm to automatically score

student essays with a minimum usage of training data. We used Doc2Vec, Word2Vec and SVD to generate the vector representation of documents. Then we used K-means clustering to cluster and further sub-cluster the documents where k is equal to five in the first iteration and k was 50 in creating sub-clusters. Finally, we propagated the actual scores of each cluster centroid to all the members of each cluster. According to our experimental results, Semantic k-means clustering using Doc2vec has performed well with an accuracy of 55% which is better than the other baseline by only labelling 15% from the given data sets.

All the experiments were conducted on the datasets with a limited size and the performance of the proposed models might increase if the experiment is replicated with relatively large datasets. In Off-topic detection, it would be highly useful to have more samples of student essays written on a particular subject but not answering the test question. Having a well-written reference answer for each topic would improve the results as well. For plagiarism detection and score propagation tasks, it would be generally better to have more data samples for each category of plagiarism and each score respectively. Therefore, the future direction in the area would be to replicate the approaches using the datasets which are collected for such a task.

Bibliography

[1] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat. Np-hardness of euclidean sum-of-squares clustering. *Machine learning*, 75(2):245–248, 2009.

[2] ASAP-SAS. Scoring short answer essays. asap short answer scoring competition system description. `http://www.kaggle.com/c/asap-sas/`, 2012.

[3] Bhagyashree Vyankatrao Barde and Anant Madhavrao Bainwad. An overview of topic modeling methods and tools. In *2017 International Conference on Intelligent Computing and Control Systems (ICICCS)*, pages 745–750. IEEE, 2017.

[4] Jerome R Bellegarda. Latent semantic mapping [information retrieval]. *IEEE Signal Processing Magazine*, 22(5):70–80, 2005.

[5] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[6] Michael W Berry, Susan T Dumais, and Gavin W O'Brien. Using linear algebra for intelligent information retrieval. *SIAM review*, 37(4):573–595, 1995.

[7] Steven Bird, Edward Loper, and Ewan Klein. Natural language toolkit. *J URL http://www. nltk. org*, 2009.

[8] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. In *Advances in neural information processing systems*, pages 601–608, 2002.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022, 2003.

[10] Roger Bradford. Techniques for processing lsi queries incorporating phrases. In *International Joint Conference on Knowledge Discovery, Knowledge Engineering, and Knowledge Management*, pages 99–117. Springer, 2014.

[11] P. Raghavan C. D. Manning and H. Schutze. *Introduction to information retrieval*. Cambridge University Press, 2008.

[12] Paul Clough and Mark Stevenson. Developing a corpus of plagiarised short answers. *Language resources and evaluation*, 45(1):5–24, 2011.

[13] Ronan Collobert and Jason Weston. A unified architecture for natural language processing: Deep neural networks with multitask learning. In *Proceedings of the 25th international conference on Machine learning*, pages 160–167. ACM, 2008.

[14] Marco Cuturi. Sinkhorn distances: Lightspeed computation of optimal transport. In *Advances in neural information processing systems*, pages 2292–2300, 2013.

[15] John R Firth. A synopsis of linguistic theory, 1930-1955. *Studies in linguistic analysis*, 1957.

[16] George W Furnas, Scott Deerwester, Susan T Dumais, Thomas K Landauer, Richard A Harshman, Lynn A Streeter, and Karen E Lochbaum. Information retrieval using a singular value decomposition model of latent semantic structure. In *Proceedings of the 11th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 465–480. ACM, 1988.

[17] Zellig S. Harris. Distributional structure. *WORD*, 10(2-3):146–162, 1954.

[18] Derrick Higgins, Jill Burstein, and Yigal Attali. Identifying off-topic student essays without topic-specific training data. *Natural Language Engineering*, 12(2):145–159, 2006.

[19] Gao Huang, Chuan Guo, Matt J Kusner, Yu Sun, Fei Sha, and Kilian Q Weinberger. Supervised word mover's distance. In *Advances in Neural Information Processing Systems*, pages 4862–4870, 2016.

[20] Anil K Jain. Data clustering: 50 years beyond k-means. *Pattern recognition letters*, 31(8):651–666, 2010.

[21] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[22] Matt Kusner, Yu Sun, Nicholas Kolkin, and Kilian Weinberger. From word embeddings to document distances. In *International conference on machine learning*, pages 957–966, 2015.

[23] Alberto Lavelli, Fabrizio Sebastiani, and Roberto Zanoli. Distributional term representations: an experimental comparison. In *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, pages 615–624. ACM, 2004.

[24] Quoc Le and Tomas Mikolov. Distributed representations of sentences and documents. In *International conference on machine learning*, pages 1188–1196, 2014.

[25] Elizaveta Levina and Peter Bickel. The earth mover's distance is the mallows distance: Some insights from statistics. In *Proceedings Eighth IEEE International Conference on Computer Vision. ICCV 2001*, volume 2, pages 251–256. IEEE, 2001.

[26] Stuart Lloyd. Least squares quantization in pcm. *IEEE transactions on information theory*, 28(2):129–137, 1982.

[27] H. P. Luhn. A statistical approach to mechanized encoding and searching of literary information. *IBM Journal of Research and Development*, 1(4):309–317, Oct 1957.

[28] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[29] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[30] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of*

the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, pages 746–751, Atlanta, Georgia, June 2013. Association for Computational Linguistics.

[31] Chandan K Reddy and Bhanukiran Vinzamuri. A survey of partitional and hierarchical clustering algorithms. In *Data Clustering*, pages 87–110. Chapman and Hall/CRC, 2018.

[32] Radim Rehurek and Petr Sojka. Software framework for topic modelling with large corpora. In *In Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks*. Citeseer, 2010.

[33] Sam T Roweis and Lawrence K Saul. Nonlinear dimensionality reduction by locally linear embedding. *science*, 290(5500):2323–2326, 2000.

[34] Yossi Rubner, Carlo Tomasi, and Leonidas J Guibas. A metric for distributions with applications to image databases. In *Sixth International Conference on Computer Vision (IEEE Cat. No. 98CH36271)*, pages 59–66. IEEE, 1998.

[35] Gerard Salton and Christopher Buckley. Term-weighting approaches in automatic text retrieval. *Information processing & management*, 24(5):513–523, 1988.

[36] Gerard Salton and Michael J. McGill. Introduction to modern information retrieval (pp. paginas 400), 1986.

[37] Stephen V Stehman. Selecting and interpreting measures of thematic classification accuracy. *Remote sensing of Environment*, 62(1):77–89, 1997.

[38] Kaveh Taghipour and Hwee Tou Ng. A neural approach to automated essay scoring. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1882–1891, 2016.

[39] Tsegaye Misikir Tashu and Tomás Horváth. Pair-wise: Automatic essay evaluation using word mover's distance. In *CSEDU (1)*, pages 59–66, 2018.

[40] Tsegaye Misikir Tashu and Tomáš Horváth. A layered approach to automatic essay evaluation using word-embedding. In *Computer Supported Education*, pages 77–94, Cham, 2019. Springer International Publishing.

[41] Tsegaye Misikir Tashu and Tomáš Horváth. Semantic-based feedback recommendation for automatic essay evaluation. In Yaxin Bi, Rahul Bhatia, and Supriya Kapoor, editors, *Intelligent Systems and Applications*, pages 334–346, Cham, 2020. Springer International Publishing.

[42] Tsegaye Misikir Tashu, Dávid Szabó, and Tomáš Horváth. Reducing annotation effort in automatic essay evaluation using locality sensitive hashing. In Andre Coy, Yugo Hayashi, and Maiga Chang, editors, *Intelligent Tutoring Systems*, pages 186–192, Cham, 2019. Springer International Publishing.

[43] Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

[44] Xiaojun Wan and Yuxin Peng. The earth mover's distance as a semantic measure for document similarity. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 301–302. ACM, 2005.