IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received January 21, 2021, accepted February 17, 2021, date of publication February 22, 2021, date of current version March 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3060940

Automatic Exam Correction Framework (AECF) for the MCQs, Essays, and Equations Matching

HOSSAM MAGDY BALAHA AND MAHMOUD M. SAAFAN
Computers and Systems Engineering Department, Faculty of Engineering, Mansoura University, Mansoura 35516, Egypt
Corresponding author: Hossam Magdy Balaha (hossam.m.balaha@mans.edu.eg)

ABSTRACT Automatic grading requires the adaption of the latest technologies. It has become essential especially when most of the courses became online courses (MOOCs). The objectives of the current work are (1) Reviewing the literature on the text semantic similarity and automatic exam correction systems, (2) Proposing an automatic exam correction framework (HMB-AECF) for MCQs, essays, and equations that is abstracted into five layers, (3) Suggesting equations similarity checker algorithm named ''HMB-MMS-EMA'', (4) Presenting an expression matching dataset named ''HMB-EMD-v1'', (5) Comparing the different approaches to convert textual data into numerical data (Word2Vec, FastText, Glove, and Universal Sentence Encoder (USE)) using three well-known Python packages (Gensim, SpaCy, and NLTK), and (6) Comparing the proposed equations similarity checker algorithm (HMB-MMS-EMA) with a Python package (SymPy) on the proposed dataset (HMB-EMD-v1). Eight experiments were performed on the Quora Questions Pairs and the UNT Computer Science Short Answer datasets. The best-achieved highest accuracy in the first four experiments was 77.95% without fine-tuning the pre-trained models by the USE. The best-achieved lowest root mean square error (RMSE) in the second four experiments was 1.09 without fine-tuning the used pre-trained models by the USE. The proposed equations similarity checker algorithm (HMB-MMS-EMA) reported 100% accuracy over the SymPy Python package which reported 71.33% only on ''HMB-EMD-v1''.

INDEX TERMS Automatic exam correction, document embedding, expression trees, MCQ matching, word embedding.

I. INTRODUCTION

Automatic grading is an approach that requires the adaption of the latest technologies. Automatic grading has become very necessary especially when most of the courses became online courses (MOOCs). This section introduces the semantic text (essays), multiple-choice questions (MCQs), and equations (expression) matching in detail as the proposed approach and suggested framework depend mainly on them.

A. SEMANTIC TEXT MATCHING

Semantic text matching is the process of finding the similarity percentage between two texts semantically [1]. They pass through (1) text pre-processing, (2) tokenization (i.e. the text can be used as a whole or tokenized into words), (3) feature extraction, and (4) the semantic similarity score is calculated between them. The process is summarized in Figure 1 and discussed in the following sub-sections.

The associate editor coordinating the review of this manuscript and approving it for publication was Feng Shao.

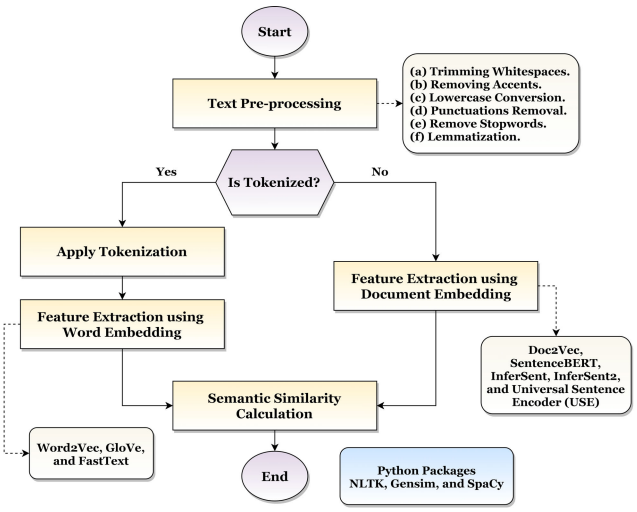

FIGURE 1. Semantic text matching steps summarization.

1) PRE-PROCESSING THE TEXT PHASE

The pre-processing phase consists of a set of steps: (1) trimming (removing) the text whitespaces from the left and right

TABLE 1. Stemming vs. Lemmatization.

Criteria	Stemming	Lemmatization
Processing Time	Less time	More time
Representation Meaning	May not have any meaning	Has meaning
Accuracy	Less accurate	More accurate
Approach	Rule-based	Dictionary-based
Applications	Preferred when the meaning is not important	Recommended when the meaning is important
Example	"Change", "Changing", "Changes", "Changed", and "Changer" are converted into "Chang"	"Change", "Changing", "Changes", "Changed", and "Changer" are converted into "Change"

such as spaces, tabs, and new line feeds, (2) removing accents from the text, (3) converting the whole text into lowercase, (4) removing the punctuations wisely, (5) removing stop-words, and (6) applying lemmatization. Any of these steps can be bypassed according to the application.

Removing stop-words is the process of filtering the text from a set of common words such as "the" and "an" [2]. Lemmatization is the process of reducing the words as it takes into account the language full vocabulary to apply the morphological analysis to words [3]. Lemmatization has another alternative is called stemming [4]. The main target of both is to reduce the inflectional forms and derivationally related forms of a word to a common base form [5]. The differences between them are shown in Table 1 [6].

2) TEXT TOKENIZATION PHASE

Tokenization is converting (breaking) the text into components, pieces (words and sub-words), punctuations, and so on [7]. The breaking is performed using a delimiter which is the "space" commonly. The tokenization algorithm should perform a check on each tokenized element and answer the following question "Should this element be tokenized or not?". For example, "U.S.A" should not be tokenized while "the Earth. It has" should be tokenized. Abbreviations, emails, and website links should be added to the exceptions also. This phase can be bypassed if we want to work with the whole text or paragraph [8].

3) EXTRACTING FEATURES FROM TEXT PHASE

The raw textual data cannot be used directly with the algorithms as most of them expect numerical data (feature vectors) [9]. They also should be in a fixed-size format rather than a variable-length text. There are different approaches to extract features from the text such as count vectorizer, term frequency (TF), term frequency-inverse document frequency (TF-IDF), word embedding, and document (paragraph) embedding [10]–[12]. Count vectorizer counts the occurrences of each word in the document to map that text into a number. It has a critical drawback that larger documents will have higher average count values than shorter ones although they may focus on the same topics [13].

The TF avoids the cons. of the count vectorizer as it divides the number of occurrences of each word in the document by the total number of words in the document. It can be considered as a normalization step. Its drawback is that it is not enough separately. The word "the" may be used more frequently in the document and hence it will give less weightage to the more meaningful words such as "school" and "university" (as they might be occurring less frequently in the document compared to the "the" word). For example, "the boy went to the school by the yellow bus.". The "the" word occurred three times while the words "boy", "school", and "bus" occurred only once [14].

The TF-IDF is a refinement on the top of the TF as it downscales the weights for the words that occur in many documents in the corpus and are therefore less informative compared to those that occur only in a smaller portion of the corpus and are more informative. The term IDF is a logarithmically scaled inverse fraction of the documents that contain the term while the term TF-IDF can be considered as the combination of the count vectorizer, TF, and IDF factors. In short words, the TF-IDF is how important a word in the document [15]. Equation 1 shows how to calculate the TF-IDF.

$$\begin{aligned} Value_{TFIDF} &= TF(t,d) \times IDF(t,D) \\ &= TF(t,d) \times \log\left(\frac{N}{\{d \in D : t \in D\}}\right) \end{aligned} \quad (1)$$

where t is the term, d is the current document, D is the documents in the corpus, N is the number of total documents in the corpus, and $\{d \in D : t \in D\}$ is the count of documents in the corpus that contains the term t.

Word embedding is a mathematical description of the individual words such that words that appear frequently in the language will have similar values which leads to derive the context [16]. For example, "lion" is closer to "cat". Word2Vec [17], GloVe [18], and FastText [19] are the common flavors of word embeddings. Word embedding has some pros. such as dimensionality reduction and performance-boosting.

Word2Vec is a shallow two-layered neural network to perform word embeddings. It groups the vector of similar words in the vector space and it can be used to compare the whole document as it takes the advantage of individual token vectors. It can be obtained using two variants: (1) Skip Gram and (2) Common Bag of Words (CBOW) [20].

Skip-gram takes the target word and tries to predict the surrounded content words, while CBOW takes a set of content words and tries to predict a target word. CBOW is higher in speed than skip-gram and provides a better frequency while skip-gram needs a small number of training records and represents even rare and uncommon words [21], [22].

Word2Vec performs word embeddings by relating target words to their context. It ignores whether some context words appear more frequently than others. For Word2Vec, a frequent co-occurrence of words creates more training data only and has no additional information. In contrast, GloVe (Global

TABLE 2. Word2Vec, GloVe, and FastText comparison.

Word2Vec	GloVe	FastText
Performs word embeddings by relating target words to their context.	A word vector combination relates directly to the probability of these words' co-occurrence in the corpus.	Enhances Word2Vec. Generalization is available as new words have the same characters as known ones.
Obtained using two variants: "Skip-Gram" and "CBOW".	Its embeddings can be interpreted as a result of the training corpus with low dimensionality that reflects the different co-occurrences.	Less training data is required since much more information can be extracted from each part of the text.
Built by Google.	Built by Standford.	Built by Facebook.

Vectors) focuses on that, the frequency of co-occurrences is critical and should not be neglected. GloVe builds the word embeddings by relating a combination of word vectors directly to the probability of these words' co-occurrence in the corpus [23], [24].

Word2Vec has a drawback that, it does not generalize to unknown words. FastText overcomes this drawback. The word embeddings from FastText look very similar to the ones generated by Word2Vec. FastText is not calculated directly, but there is a combination of lower-level embeddings. There are two main advantages: (1) generalization is available as new words have the same characters as known ones and (2) less training data is required since much more information can be extracted from each part of the text [25]. Table 2 summarizes Word2Vec, GloVe, and FastText. The pre-trained model weights can be downloaded and used for Word2Vec, GloVe, and FastText (from their official website) https://code.google.com/archive/p/word2vec/, https://nlp.stanford.edu/projects/glove/, and https://fasttext.cc/ respectively.

Document (i.e. paragraph or sentence) embedding considers the entire sentence and their semantic information represented as vectors. This will help machines in understanding the context of the entire text. There are many techniques including the state-of-the-art ones such as Doc2Vec, SentenceBERT, InferSent, InferSent2, and Universal Sentence Encoder (USE). Doc2Vec is an extension of the Word2Vec. It is one of the most popular techniques that was used as an unsupervised algorithm above the Word2Vec. There are two ways to achieve that: (1) Distributed Memory version of Paragraph Vector (PVDM) and (2) Distributed Bag of Words version of Paragraph Vector (PVDOBW). Both of them are useful but the first is recommended for most of the tasks.

BERT is a Bidirectional Encoder Representations from Transformers for the pre-training over a lot of unlabeled textual data. It is based on Google's released transformer architecture. It is working mechanism is (1) Pre-training: it learns the language representation from a huge amount of unlabeled data in an unsupervised manner and (2) The pre-trained model can be fine-tuned after that in a supervised manner using a small amount of labeled trained data. It is used in fine-tuning the language representation in different machine learning tasks.

XLNet is a large bidirectional transformer that uses the improved training methodology, larger data, and more computational power to achieve better performance metrics than BERT on twenty language tasks. To improve the XLNet training, it applied the permutation language modeling where all of the tokens were predicted in random order. This helped the model to learn from the bidirectional relationships. Hence, it achieved better dependencies and relations between the words. It was trained with over 130 Gigabytes of the textual data for 2.5 days using 512 tensor processing unit (TPU) chips running which was much larger than the original BERT [26].

RoBERTa (Robustly optimized BERT approach) was introduced by Facebook. It is a retraining of the BERT with an improved training methodology using 1000% more data and computational power. RoBERTa removed the Next Sentence Prediction (NSP) task from BERT's pre-training and introduced dynamic masking [27]. It used 160 Gigabytes of the textual data for pre-training including (1) 16 Gigabytes of Books Corpus and English Wikipedia used in BERT, (2) The CommonCrawl News dataset, (3) web text corpus, and (4) stories from the Common Crawl [27].

SentenceBERT is a (BERT)-based model that has four key concepts: (1) Attention, (2) Transformers, (3) BERT, and (4) the Siamese network. The latter one is used to calculate the cosine similarity of two-pair sentences [28], [29]. InferSent is a supervised sentence embedding technique. It is trained on the Stanford Natural Language Inference (SNLI) dataset. It contains $570K$ human-generated English sentence pairs and uses the GloVe vectors for the pre-trained word embedding. InferSent2 is similar to InferSent but uses the FastText pre-trained vectors [30]. The USE is useful for multi-task learning which is useful in sentence similarity and text classification. Its encoder is based on two encoder models and anyone of them can be used. They are (1) Transformer Encoder and (2) Deep Averaging Network (DAN) Encoder [31], [32].

GPT-3 (Generative Pre-trained Transformer 3) is arguably the most powerful member in the family of the NLP models including BERT and GPT-series. GPT-3 is a deep-neural-network-powered language model that was developed by OpenAI. Words or phrases in the GPT-3 were randomly removed from the text and the model learned to fill-in them using only the context (i.e. surrounding words). It removes the necessity of the traditional models fine-tuning for each NLP task. It contains 175 billion parameters and was trained on the Common Crawl dataset. It resulted in a powerful and generalizable model [33].

4) CALCULATING THE SEMANTIC SIMILARITY PHASE

Semantic similarity is a metric defined over a set of documents or terms when the idea of the distance between items is based on the similarity of their meaning or semantic content. It includes ''is a'' relations [34]. For example, the ''car'' word

can be treated similarly to the "bus" word. There are different methods such as (1) Cosine similarity, (2) Jaccard similarity, (3) Manhattan distance, and (4) Euclidian distance [35].

Cosine similarity (Equation 2) measures how similar the documents are in size irrespective. It calculates the cosine angle between two vectors projected in the multi-dimensional space. The smaller the angle, the higher the cosine similarity [36]. Cosine similarity is comparing two real-valued vectors while Jaccard similarity (Equation 3) is for comparing two binary vectors or sets [37]. The Euclidean distance (Equation 4) is a straight-line distance between two vectors [38]. The Manhattan distance (Equation 5) is considered a way to compute the distance between two points when you are not able to get the straight-line [39].

$$Similarity_{Cosine} = \frac{A \bullet B}{\| A \| \bullet \| B \|} \tag{2}$$

$$Similarity_{Jaccard} = \frac{| A \cap B |}{| A \cup B |} = \frac{| A \cap B |}{| A | + | B | - | A \cap B |} \tag{3}$$

$$Distance_{Euclidean} = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2} \tag{4}$$

$$Distance_{Manhattan} = \sum_{i=1}^{n} | A_i - B_i | \tag{5}$$

where A and B are the numerical vectors, and n is the number of elements.

5) SOFTWARE PACKAGES

There are multiple software packages with Python language (the used programming language in the current research) such as Gensim [40], SpaCy [41], and NLTK [42]. Gensim is a Python package for topic modeling, document indexing, and similarity retrieval with large corpora [43]. SpaCy is a free open-source package for natural language processing (NLP) in Python. It is built on the very latest state-of-the-art researches. It comes with a set of pre-trained statistical models and word vectors. It supports tokenization for more than 60 languages [44]. NLTK is a leading platform for building different Python applications to work with human language data. It provides easy-to-use interfaces and lexical resources such as WordNet. It has a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning [45].

B. MCQ MATCHING

There are different types of multiple-choice questions (MCQs) such as single select, multiple select, and dropdown MCQs. The correction of them is straightforward but depends on the type of the MCQ itself. It checks directly if the student answer is similar to the reference answer or not. It extracts the matching percentage and based on it and the correction criteria, the mark is put. For example, for multiple select MCQ, if there are five selections (three are correct) and the mark is one, then for each selection, it is 0.2. Accumulation is put on each correct selection (if the selection should be selected and the student selected it or if the selection should not be selected and the student did not select it).

C. EQUATIONS MATCHING

An equation in its simplest form is the combination of symbols, functions, numbers, and operators. For example $(a+b)*c$. Equations (or Expressions) matching is the process of determining if two equations (or expressions) lead to the same result [46]. There are three representations of expressions: (1) prefix notation, (2) infix notation, and (3) postfix notation [47]. If the operators appear between the operands then it is the infix notation. If the operators appear before and after the operands then they refer to prefix and postfix notations respectively. The following inner sub-sections present the expression trees, how to convert from the infix to prefix notation, and how to build the expression trees recursively. The proposed equations matching algorithm (discussed in a later section) depends mainly on them.

1) EXPRESSION TREES

Expression trees, as shown in Figure 2, are binary trees in which each internal node presents an operator while the leaves are the operands. They are useful in many applications such as equations (or expressions) matching [48], [49]. To construct an expression tree, the expression is converted to the infix notation, and then the tree is built [50].

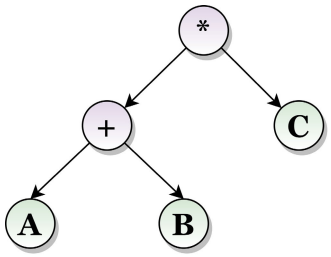

FIGURE 2. Example on the expression trees.

2) INFIX TO PREFIX NOTATION CONVERSION

The following subsection shows how to convert from the infix notation to the prefix notation. It is limited to integer positive numbers and $*$, $-$, $/$, and $+$ operators. The input to the algorithm is the infix expression while the output is the prefix expression. The steps are summarized in Algorithm 1 and discussed below.

1) Read the infix expression.
2) Get the elements from infix expression using the regular expression "[a−zA−Z]+[0−9]∗|[0−9]+|[(|)|+|−|∗|/]" [51]. This regex will extract the variables, numbers, and operators.
3) Initiate a "reverse" variable.
4) For each element in the elements array: (1) exchange "(" with ")" and vice versa, (2) encapsulate the

Algorithm 1 Infix to Prefix Notation Conversion Pseudocode

```
function INFIX2PREFIX(infix)  \\ The infix to prefix notation conversion function. It accepts the the infix notation and returns the prefix notation.
    re ← "[a-zA-Z]+[0-9]*|[0-9]+| [(|)|+|-|*|/]"  \\ The used regular expression to extract variables, numbers, and operators.
    elements ← Regex(infix, re)  \\ The extracted variables, numbers, and operators.
    reverse ← ""  \\ Initiate the "reverse" variable.
    for each element ← elements do  \\ Iterate on each element in the elements array.
        element ← Exchange(element, "(", ")")  \\ Exchange between "(" and ")" and vice versa in the element variable.
        element ← EncapsulateIfOperand(element)  \\ Encapsulate the operands with curly braces.
        reverse ← element + reverse  \\ Apply the reversing operation.
    rem ← "{[a-zA-Z]+[0-9]*}|{[0-9]+}|[(|)|+|-|*|/]"  \\ The used regular expression to extract the modified operands and operators.
    modifiedElements ← Regex(reverse, rem)  \\ The extracted modified operands and operators.
    prefix ← ""  \\ Initiate the "prefix" variable.
    stack ← Stack()  \\ Initiate the "stack" variable.
    for each modifiedElement ← modifiedElements do  \\ Iterate on each modified element in the modifiedElements array.
        prefix ← AppendIfOperand(prefix, modifiedElement)  \\ Append the element the "prefix" variable if it is a modified operand.
        stack ← PushIfLeftParenthesis(stack, modifiedElement)  \\ Push the element into the stack if it is a left parenthesis "(".
        stack, prefix ← HandleIfOperator(stack, prefix, modifiedElement)  \\ If the element is an operator, pop the top higher-precedence stack elements, append them to the "prefix" variable, and push the element after that into the stack.
        stack, prefix ← HandleIfRightParenthesis(stack, prefix, modifiedElement)  \\ If the element is a right parenthesis ")", pop the top stack elements until it reaches a left parenthesis "(" and append them to the "prefix" variable (neglect the parentheses).
    stack, prefix ← HandleRemaining(stack, prefix)  \\ If there any remaining elements in the stack, pop them and append them to the "prefix" variable (neglect the parentheses).
    modifiedElements ← Regex(prefix, rem)  \\ Get the modified elements but now from the "prefix" variable.
    reverse ← ""  \\ Initiate the "reverse" variable.
    for each modifiedElement ← modifiedElements do  \\ Iterate on each modified element in the modifiedElements array.
        reverse ← modifiedElement + reverse  \\ Apply the reversing operation.
    prefix ← reverse
    return prefix  \\ Return the required prefix notation.
```

operands (variables and numbers) with curly braces, and (3) perform the reversing on the form "reverse = element + reverse" (i.e. each incoming element is put at the beginning of the "reverse" string).

5) Get the modified elements from the reversed expression using the regular expression "{[a − zA − Z] + [0 − 9]∗}|{[0−9]+}|[(|)|+|−|∗|/]". This regex will extract the modified operands and operators.
6) Initiate a "prefix" variable and a stack.
7) For each element in the modified elements array: (1) if the element is a modified operand, append to the "prefix" variable, (2) if the element is a left parenthesis "(", push it into the stack, (3) if the element is an operator, pop the top higher-precedence stack elements, append them to the "prefix" variable, and push the element after that into the stack, and (4) if the element is a right parenthesis ")", pop the top stack elements until it reaches a left parenthesis "(" and append them to the "prefix" variable (neglect the parentheses).
8) If there any remaining elements in the stack after iterating on all modified elements, pop them and append them to the "prefix" variable (neglect the parentheses).
9) Similar to the fifth step, get the modified elements from the "prefix" variable and re-reverse the elements on the form "reverse = element + reverse".

3) BUILDING THE PREFIX EXPRESSION TREE

The following subsection shows how to build the prefix expression tree. The algorithm depends on the recursion [52]. The input to the algorithm is the prefix expression while the output is the prefix expression tree. The steps are summarized in Algorithm 2 and discussed below.

1) Read the prefix expression.
2) Get the first modified element from the expression.

Algorithm 2 Building the Prefix Expression Tree Pseudocode

```
function BUILDEXPRESSIONTREE(prefix)  \\ The function accepts the prefix expression and returns the prefix expression tree.
    firstElement ← PopFirstModifiedElement(prefix)   \\ Pop the first modified element from the expression.
    node ← Node(firstElement)                        \\ Create a node object.
    if IsOperandNode(node) then                      \\ Check if the node object is an operand.
        return node                                  \\ If TRUE, return the node object.
    node.Left ← BuildExpressionTree(prefix)          \\ Apply recursion on the rest of the expression to get the left branch.
    node.Right ← BuildExpressionTree(prefix)         \\ Apply recursion on the rest of the expression to get the right branch.
    return node                                      \\ Return the required prefix expression tree.
```

3) Create a node object. A node object is an object from the ''Node'' class that contains the data, left branch, and right branch members. The left and right branches are ''NULL'' by default.
4) If the first modified element (i.e. node object) is an operand, return the node object. Otherwise, continue the next steps.
5) Perform recursion (repeat the second step) on the rest of the expression to get the left branch.
6) Perform recursion (repeat the second step) on the rest of the expression to get the right branch.
7) Return the node object.

4) SOFTWARE PACKAGES

There are multiple software packages with Python language (the used programming language in the current research) that handles the expressions such as SymPy [53]. SymPy is a package for symbolic mathematics. It aims to become a full-featured computer algebra system (CAS). It keeps the code as simple as possible to be comprehensible and easily extensible.

From this basic introduction, the current study works on the automatic grading approach using the latest technologies. This will be applied to essays, MCQs, and equations (i.e. expressions). The main contributions of the current study are:

1) Proposing an automatic exam correction framework (HMB-AECF) for MCQs, essays, and equations.
2) Comparing the different approaches to convert textual data into numerical data (Word2Vec, FastText, Glove, and USE) using three well-known Python packages (Gensim, SpaCy, and NLTK).
3) Applying different experiments on the ''Quora Questions Pairs'' and the ''UNT Computer Science Short Answer'' datasets.
4) Suggesting equations similarity checker algorithm named ''HMB-MMS-EMA''.
5) Presenting an expression matching dataset named ''HMB-EMD-v1''.
6) Comparing the proposed equations similarity checker algorithm ''HMB-MMS-EMA'' with a well-known Python package (SymPy) on the proposed dataset ''HMB-EMD-v1''.

The rest of the paper is organized as follows: Section two reviews a set of the related works. Section three discusses the suggested framework and proposed algorithm in detail. Section four shows the performed experiments and used datasets. It presents the compiled dataset, reports the experimental results, and discusses them in detail. Finally, the conclusion of the presented work and future work are reported in Section five.

II. RELATED WORK

There are several proposed approaches and systems in the field of automatic scoring and grading. Most of them worked on the grading of essays and short answers. Pribadi *et al.* [54] developed a system for automatic short answer scoring. They compared multiple methods that applied the overlapping methods to determine the similarity degree between the references and students' answers. They showed that the cosine coefficient method reported the best results than the Dice and Jaccard coefficient methods.

Suzen *et al.* [55] focused on the short answer questions automatic grading. They reported the experimental results on their own compiled dataset. The dataset was collected and compiled from a computer science class at North Texas University. They applied several data mining techniques to the student answers corpus to calculate the similarity scores. They argued that the computational methods could be used to improve the reliability of human scoring but not replace it.

Mohler *et al.* [56] worked on the computer-assisted assessment of the short student answers. They merged the graph alignment features using the lexical-semantic similarity measures. They reported that the answers can be graded more accurately compared with if the semantic measures were used in isolation.

Hassan *et al.* [57] suggested a supervised learning algorithm for the short answer automatic scoring which was based on paragraph embeddings. They discussed significant deep learning-based models for generating paragraph embeddings. They presented a detailed choice criterion of paragraph embedding.

Mohler *et al.* [58] investigated unsupervised techniques for the automatic short answer grading problem. They compared a set of text similarity knowledge-based and corpus-based measures. They reported the effect of domain and size on the corpus-based measures. They introduced a technique to improve system performance by integrating automatic feedback from the student answers.

TABLE 3. Related works pros. and cons.

Work	Year	Pros.	Cons.
Suzen et al. [56]	2020	They worked on the short answer questions automatic grading. They compiled their dataset	They did not work on the equations nor MCQs. They did not compare their work with the pre-built embedding models
Hassan et al. [58]	2018	They approached a supervised learning algorithm and discussed deep learning based models for generating paragraph embeddings	They did not work on the equations nor MCQs. They did not purpose a general framework to follow
Pribadi et al. [55]	2017	They worked on the essay and reported a set of similarity scores	They did not work on the equations nor MCQs
Mohler et al. [57]	2011	They compiled their dataset and made it for public usage	They did not work on the equations nor MCQs
Mohler et al. [59]	2009	They approached unsupervised learning algorithms and compared a set of text similarity knowledge-based and corpus-based measures	They did not work on the equations nor MCQs. They did not compare their work with the pre-built embedding models

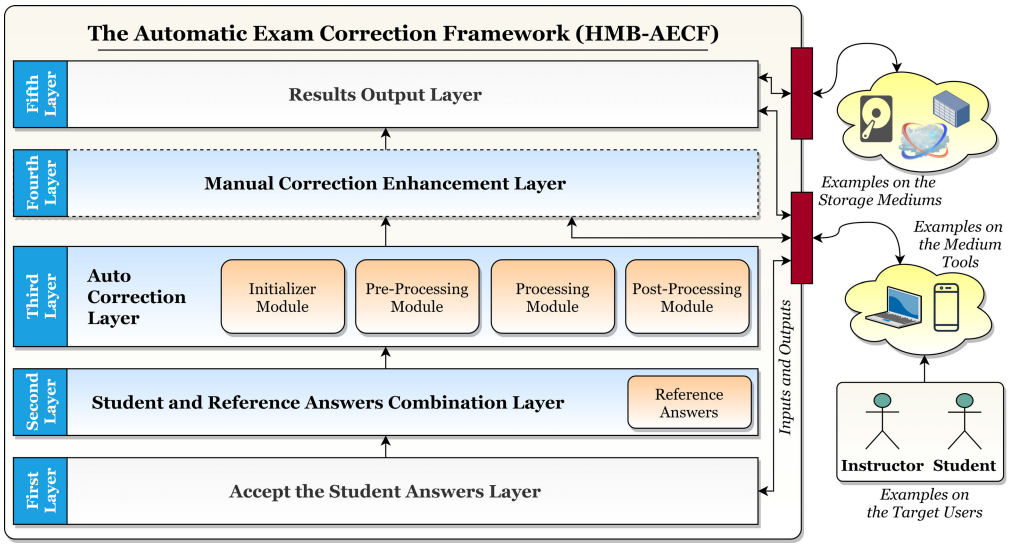

FIGURE 3. Automatic Exam Correction Proposed Framework (HMB-AECF).

Table 3 shows the pros. and cons. of the discussed related works. They are ordered from the latest to the oldest according to the publishment year.

III. AUTOMATIC EXAM CORRECTION FRAMEWORK (HMB-AECF)

The proposed automatic exam correction framework named HMB-AECF is shown in Figure 3. It consists of five layers and they are discussed in the following five sub-sections.

A. FIRST LAYER: ACCEPT THE STUDENT ANSWERS LAYER

The first layer is responsible for accepting the student's answers to different questions. The questions can be MCQs (single selection and multiple selections), essays (single step and multiple steps), or equations (single step and multiple steps). Each student answer is encapsulated in a header in the format shown in Figure 4.

From Figure 4, it consists of six parts: (1) The "Student ID" field has four bytes to hold unsigned students identifier numbers from 0 up to 2^{32}. (2) The "Question ID" field has four bytes to hold unsigned questions identifier numbers from 0 up to 2^{32}. (2) The "Question Type" field has four

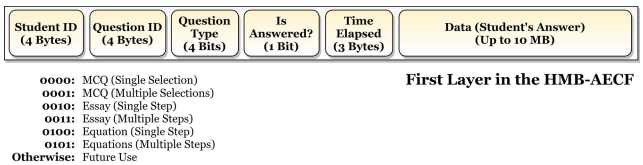

FIGURE 4. First layer: Accept the student answers layer.

bits to hold the type of question such as 0000 for MCQ (Single selection). The combinations from 0110 and above are left for future use for other question types. (3) The "Is Answered?" field has a single bit (Boolean) to indicate if the student answered the question or not. It will help to bypass the automatic correction process and set a zero mark. (4) The "Time Elapsed" field has 3 bytes to hold the student elapsed time to answer the question. It is useful also when the question is timed and has a maximum time. (5) The "Data" field is up to 10 Megabytes to hold the student's answer.

B. SECOND LAYER: STUDENT AND REFERENCE ANSWERS COMBINATION LAYER

The second layer is responsible for combining each student's answer with the corresponding reference answer.

TABLE 4. The different question types with the corresponding pre-processors, processors, and post-processors.

Code	Type	Pre-Processing	Processing	Post-Processing	Requires Threshold
0000	MCQ (Single Selection)	N/A	Direct Comparison (i.e. comparing the reference answer (selection) with the student answer (selection))	N/A	No
0001	MCQ (Multiple Selections)				
0010	Essay (Single Step) Keywords: False	Whitespaces, accents, punctuations, and stop-words removal, lowering, and lemmatization	(a) Tokenization and Word Embedding using Word2Vec, GloVe, or FastText and (b) Paragraph Embedding using Doc2Vec, SentenceBERT, InferSent, InferSent2, or USE	Calculating the Cosine similarity and Pearson correlation scores	Yes
0011	Essay (Multiple Steps) Keywords: False				
0010	Essay (Single Step) Keywords: True		Finding the exact occurrences of keywords in the text	Calculating the similarity score	No
0011	Essay (Multiple Steps) Keywords: True				
0100	Equation (Single Step)	Removing whitespaces and normalization	Prefix conversion, expression tree construction, expanding the expression, and refining the expression.	Calculating the similarity score	Yes
0101	Equations (Multiple Steps)				

The reference answer for an MCQ is the correct selection, for an essay can be keywords or the actual answer and for an equation is the reference equation. This layer encapsulates the previous layer header and produces a new header as shown in Figure 5.

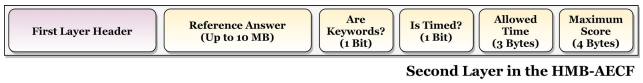

FIGURE 5. Second layer: Student and reference answers combination layer.

From Figure 5, it consists of six parts: (1) The ''First Layer Header'' field is the data from the previous layer. (2) The ''Reference Answer'' field is up to 10 Megabytes per answer to hold the reference correct answer. (3) The ''Are Keywords?'' field has a single bit (Boolean) to indicate if the reference answer is composed of keywords or the full answer. It is used only when the question type is an essay. (4) The ''Is Timed?'' field has a single bit (Boolean) to indicate if the question is timed or not. If the question is timed, the next field will be used. (5) The ''Allowed Time'' field has 3 bytes to hold the allowed time to answer the question. (6) The ''Maximum Score'' field has 4 bytes to hold the maximum score as a floating-point number.

C. THIRD LAYER: AUTO CORRECTION LAYER

The auto-correction layer consists of four modules: (1) the initializer module, (2) the pre-processing module, (3) the processing module, and (4) the post-processing module. The initializer module checks the question type (from the first layer) and determines if the question is a single step or multiple steps. To generalize the two cases using loops or queues, it encapsulates the single-step question into an array of a single element. It also determines the question main type: MCQ, essay (text), or equation to determine the corresponding pre-processor, processor, and post-processor modules as shown in Table 4.

This layer encapsulates the previous layer header and produces a new header as shown in Figure 6.

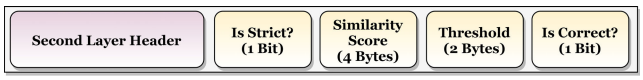

FIGURE 6. Third layer: Auto correction layer.

From Figure 6, it consists of five parts: (1) The ''Second Layer Header'' field is the data from the previous layer. (2) The ''Is Strict?'' field has a single bit (Boolean) to indicate if the essays should remove accents and punctuations and apply the lemmatization or not. It is useful when the accents and punctuations are useful and there is no necessity for lemmatization. (3) The ''Similarity Score'' field has 4 bytes to hold the similarity score as a floating-point number. (4) The ''Threshold'' field has 2 bytes to hold the threshold value used in determining if the answer is correct or not. It is used only with essays. (5) The ''Is Correct?'' field has a single bit (Boolean) to indicate if the answer is correct or not.

1) EQUATIONS MATCHING ALGORITHM (HMB-MMS-EMA)

The current sub-section discusses the suggested equations matching algorithm named (HMB-MMS-EMA) shown in Algorithm 3 and illustrated graphically in Figure 7. Algorithm 3 is discussed below.

Algorithm 3 The Suggested Equations Matching Algorithm, HMB-MMS-EMA, Pseudocode

function PREPROCESSEXPRESSION(*expression*) \\ The function accepts the expression and returns the pre-processed expression.
    trimmedExpression ← RemoveWhitespaces(*expression*) \\ Remove the whitespaces from the expression.
    normExpression, *lookupTable* ← Normalize(*trimmedExpression*) \\ Normalize the expression and get the normalized expression and the lookup table.
    return *normExpression*, *lookupTable* \\ Return the pre-processed expression with the lookup table.
function PROCESSEXPRESSION(*expression*, *lookupTable*) \\ The function accepts the expression and the merged lookup table. It returns the processed expression tree.
    replaced ← Replace(*expression*) \\ The square brackets are replaced with parentheses in the expression.
    prefix ← Infix2Prefix(*replaced*) \\ The prefix notation is extracted.
    expressionTree ← BuildExpressionTree(*prefix*) \\ The expression tree is built.
    expandedTree ← ExpandExpressionTree(*expressionTree*) \\ The expression tree is expanded.
    refined ← RefineTree(*expandedTree*) \\ The expanded tree is refined.
    restored ← RestoreExpression(*refined*, *lookupTable*) \\ The refined expression is restored using the lookup table.
    cleaned ← RemoveCurlyBraces(*restored*) \\ Remove the curly braces from restored expression.
    result ← Stringify(*cleaned*) \\ Stringify the cleaned expression.
    return *result* \\ Return the processed and expanded expression tree.
function POSTPROCESSEXPRESSION(*first*, *second*) \\ The function accepts the two processed expressions and returns the similarity score.
    similarity ← CalculateSimilarity(*first*, *second*) \\ Calculate the similarity score between the two expressions.
    return *similarity* \\ Return the similarity score.
function HMB-MMS-EMA(*firstExpression*, *secondExpression*) \\ The function accepts the two expressions and returns the similarity between them.
    firstPreprocessed, *firstLookupTable* ← PreprocessExpression(*firstExpression*) \\ Pre-process the first expression.
    secondPreprocessed, *secondLookupTable* ← PreprocessExpression(*secondExpression*) \\ Pre-process the second expression.
    lookupTable ← MergeLookupTables(*firstLookupTable*, *secondLookupTable*) \\ Merge the two lookup tables.
    firstProcessed ← ProcessExpression(*firstPreprocessed*, *lookupTable*) \\ Process the first pre-processed expression.
    secondProcessed ← ProcessExpression(*secondPreprocessed*, *lookupTable*) \\ Process the second pre-processed expression.
    similarity ← PostprocessExpression(*firstProcessed*, *secondProcessed*) \\ Post-process the two processed expressions.
    return *similarity* \\ Return the required similarity score.

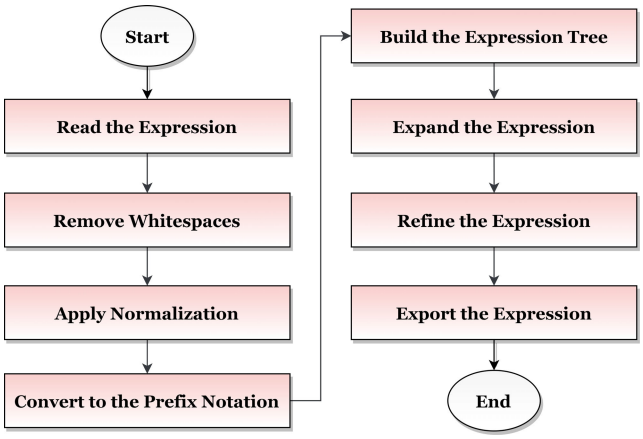

FIGURE 7. Graphical illustration of the Equations Matching Algorithm (HMB-MMS-EMA).

The HMB-MMS-EMA starts by reading the two expressions required to find the similarity between them. The following steps (i.e. following the three phases in Table 4) are applied to each expression until the similarity score is calculated between them:

1) The expression is read.
2) The pre-processing is applied to the expression by removing the whitespaces and applying normalization. In the normalization step, the variables take the annotation "v" followed by a number. For example, "(a + b)*c" should be normalized to "(v1 + v2)*v2".
3) The variables with their normalized annotations are stored in a lookup table. All lookup tables should be merged together. It will be used in a later step.
4) The square brackets used with expressions are replaced with parentheses.
5) The expression is converted into the prefix notation and the expression tree is built after that (discussed previously in Algorithm 1 and Algorithm 2).
6) The expression tree expansion is generated as follows in the next inner-steps.
    a) Read the expression tree.

b) Read the next node element (initially is the root node).
c) Perform recursion (i.e. repeat the second inner-step) on the left node if it exists.
d) Perform recursion (i.e. repeat the second inner-step) on the right node if it exists.
e) Check if the current node data is a modified operand, encapsulates the data into an array.
f) If the current node data is a plus "+" or minus "−", update the data to be on the form "left_node_data + [operator] + right_node_data".
g) If the current node data is an asterisk "*", perform a multiplication spread. Multiplication spread is performed when the left branch is multiplied to the right branch. It generates $W \times M$ operations where W and M are the left and right branch elements counts respectively. For example, the expressions"(a + b) * (c + d)" should be spread into "a * c + a * b + b * c + b * d".
h) If the current node data is a forward slash "/", perform a division spread. The division spread is performed when the left branch is divided by the right branch. It generates W operations. For example, the expression"(a + b) / (c + d)" should be spread into "a / (c + d) + b / (c + d)".
i) The expanded expression is refined after that from the same divisions, zeros, and ones. The zeros with their operators are removed if the operator is a plus "+" or minus "−". The zero's portion is removed if the operator is an asterisk "*".
j) The ones with their operators are removed if the operator is an asterisk "*" or a forward slash "/". For example, "a * 0 + a + 0 + a * 1 + 1 * 0 + 1 * 1" should become "a + a + a + 1".
k) It is refined also if there are divisions on the same operands. For example, "b / b" should become "1".
l) The refined expression is exported. This step restores the normalized annotations to their original variables using the built lookup table. It removes the curly braces also.
m) The stringified expression is returned as the processing output.

7) Calculate the similarity score between the two processed expressions and return it as the output of the algorithm.

Table 5 shows an example of how the suggested algorithm works.

D. FOURTH LAYER: MANUAL CORRECTION ENHANCEMENT LAYER

This layer is optional and can be bypassed. The target of this layer is to refine (i.e. enhance) the automated reported results. The instructor should pass on each student's answer (with its correction) anonymously to approve or update the result manually. This layer encapsulates the previous layer header and produces a new header as shown in Figure 8.

TABLE 5. Example on the steps of the suggested HMB-MMS-EMA.

Step	Output
Read the Expression	(x1 + y) * (a + q) / 5 + 1 * 0
Remove Whitespaces	(x1+y)*(a+q)/5+1*0
Apply Normalization	(v1+v2)*(v3+v4)/5+1*0
Convert to Prefix Notation	+*+{v1}{v2}/+{v3}{v4}{5}*{1}{0}
Build the Expression Tree	(+ (* (+ [{v1}] [{v2}]) (/ (+ [{v3}] [{v4}]) [{5}])) (* [{1}] [{0}]))
Expand the Expression	['{v1}', '*', '{v3}', '/', '{5}', '+', '{v1}', '*', '{v4}', '/', '{5}', '+', '{v2}', '*', '{v3}', '/', '{5}', '+', '{v2}', '*', '{v4}', '/', '{5}', '+', '{1}', '*', '{0}']
Refine the Expression	['{v1}', '*', '{v3}', '/', '{5}', '+', '{v1}', '*', '{v4}', '/', '{5}', '+', '{v2}', '*', '{v3}', '/', '{5}', '+', '{v2}', '*', '{v4}', '/', '{5}']
Export the Expression	x1*a/5+x1*q/5+y*a/5+y*q/5

FIGURE 8. Fourth layer: Manual correction enhancement layer.

From Figure 8, it consists of four parts: (1) The "Third Layer Header" field is the data from the previous layer. (2) The "Is Approved?" field has a single bit (Boolean) to indicate if the original score generated by the framework is approved or not. If the original score is approved, the "Updated Score" field will be empty. (3) The "Original Score" field has 4 bytes to hold the original achieved score by the framework as a floating-point number. (4) The "Updated Score" field has 4 bytes to hold the updated score as a floating-point number.

E. FIFTH LAYER: RESULTS OUTPUT LAYER

This layer is responsible for storing the results in the storage medium (i.e. database or cloud storage) and displaying the final report to the user. It does not generate new headers.

IV. EXPERIMENTS AND DISCUSSION

The current section discusses the applied experiments and reports the corresponding results. It is divided into two types of experiments. They are (1) Text Matching Experiments and (2) Expressions Matching Experiments.

A. TEXT MATCHING EXPERIMENTS

The Quora Questions Pairs dataset [59] is used in the first 4 experiments. It consists of 404, 290 records (149, 263 duplicated and 255, 027 non-duplicated) of potential question duplicate pairs. Each record has a unique identifier, full text for two question pairs, and a Boolean value that indicates whether these pairs are duplicates or not.

The UNT Computer Science Short Answer dataset [60] is used in the next 4 experiments. It consists of 2, 442 records

TABLE 6. The experiments 1 to 8 configurations.

Experiment	Dataset	Approach	Is Strict	Accents Removal	Punctuations Removal	Lemmatization	Stop-words Removal	Whitespace Removal	Tokenization
1	Quora Questions Pairs [60]	Word Embedding	✗	✓	✓	✓	✓	✓	✓
2			✓	✗	✗	✗	✗	✓	✓
3		Paragraph Embedding	✗	✓	✓	✓	✓	✓	✗
4			✓	✗	✗	✗	✗	✓	✗
5	UNT Computer Science Short Answer [61]	Word Embedding	✗	✓	✓	✓	✓	✓	✓
6			✓	✗	✗	✗	✗	✓	✓
7		Paragraph Embedding	✗	✓	✓	✓	✓	✓	✗
8			✓	✗	✗	✗	✗	✓	✗

TABLE 7. Experiment 1 using Word2Vec (GoogleNews-vectors-negative300.bin).

Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
		Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
Gensim	Sum	66.10%	66.09%	66.76%	66.85%	66.86%	66.85%
	Mean	66.10%	66.09%	66.76%	66.85%	66.86%	66.85%
SpaCy	Sum	66.39%	66.39%	67.07%	67.09%	67.10%	67.09%
	Mean	66.39%	66.39%	67.07%	67.09%	67.10%	67.09%
NTLK	Sum	65.76%	65.76%	66.73%	67.09%	**67.11%**	67.09%
	Mean	65.76%	65.76%	66.73%	67.09%	**67.11%**	67.09%

of 10 assignments with between 4 and 7 questions each and 2 exams with 10 questions each. These assignments (or exams) were assigned to an introductory computer science class at the University of North Texas. The data is in plaintext format. Each assignment includes the question, instructor answer, and set of student answers with the average grades of 2 annotators included. The scores were normalized to the scale [0 : 5] before being averaged.

Table 6 summarizes the different configurations of the first eight experiments.

Experiment 1*:* In the pre-processing phase, whitespace, punctuations, stop-words, and accents are removed and lemmatization is applied (the "Is Strict?" flag is set to False in the third layer header). The tokenization phase is applied and the text is converted into words. The overall text vector is calculated using the sum and mean (average) of the word vectors. The pre-trained models of Word2Vec (i.e. GoogleNews-vectors-negative300.bin), GloVe (i.e. glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt), and FastText (i.e. wiki-news-300d-1M.vec and crawl-300d-2M.vec) are used in the word embedding phase. The cosine similarity and Pearson correlation are calculated. The 95%, 90%, and 85% thresholds (if above, it is duplicate, and otherwise, it is not duplicate) are set to determine the number of matches. The accuracy is calculated by dividing the number of correct matches by the actual number of records. The results are captured in Table 7, Table 8, and Table 9 for Word2Vec, GloVe, and FastText respectively.

From Experiment 1, the GoogleNews Word2Vec pre-trained model (Table 7) reports that all of the accuracies are in the range of 65% to 68%. The best accuracy (67.11%) is achieved using the NLTK package with the 85% threshold and Cosine similarity score. Figure 9 shows the confusion matrix of the best score record. The overall vector calculation using the sum and mean reported the same results which indicate that the cosine similarity and Pearson correlation are not affected by that.

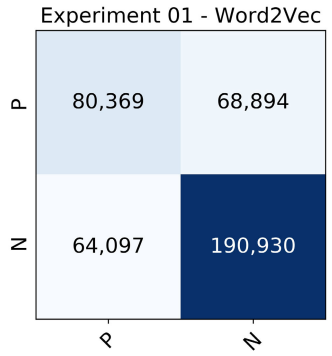

FIGURE 9. Experiment 1 Word2Vec confusion matrix.

The GloVe 6B pre-trained model (Table 8) reports that all of the accuracies are in the range of 66% to 68%. The best accuracy (67.87%) is achieved using the SpaCy package with the 90% threshold and Cosine similarity score. Figure 10 shows the confusion matrix of the best score record. The GloVe 42B pre-trained model (Table 8) reports that all of the accuracies are in the range of 59% to 68%. The best accuracy (67.58%) is achieved using the SpaCy package with the 95% threshold and Cosine similarity score. Figure 11 shows the confusion matrix of the best score record. The GloVe 840B

TABLE 8. Experiment 1 using GloVe (glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt).

Pretrained Model	Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
			Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
glove.6B.300d.txt	Gensim	Sum	66.54%	66.53%	67.34%	66.92%	66.93%	66.92%
		Mean	66.54%	66.53%	67.34%	66.92%	66.93%	66.92%
	SpaCy	Sum	67.06%	67.05%	**67.87%**	67.46%	67.47%	67.46%
		Mean	67.06%	67.05%	**67.87%**	67.46%	67.47%	67.46%
	NTLK	Sum	66.31%	66.30%	67.50%	66.94%	66.93%	66.94%
		Mean	66.31%	66.30%	67.50%	66.94%	66.93%	66.94%
glove.42B.300d.txt	Gensim	Sum	67.08%	67.07%	66.50%	61.22%	61.15%	61.22%
		Mean	67.08%	67.07%	66.50%	61.22%	61.15%	61.22%
	SpaCy	Sum	**67.58%**	67.56%	66.84%	61.02%	60.94%	61.02%
		Mean	**67.58%**	67.56%	66.84%	61.02%	60.94%	61.02%
	NTLK	Sum	67.16%	67.16%	65.89%	59.62%	59.53%	59.62%
		Mean	67.16%	67.16%	65.89%	59.62%	59.53%	59.62%
glove.840B.300d.txt	Gensim	Sum	66.93%	66.93%	67.57%	65.26%	65.27%	65.26%
		Mean	66.93%	66.93%	67.57%	65.26%	65.27%	65.26%
	SpaCy	Sum	67.43%	67.44%	**68.09%**	65.72%	65.73%	65.72%
		Mean	67.43%	67.44%	**68.09%**	65.72%	65.73%	65.72%
	NTLK	Sum	66.90%	66.90%	67.75%	64.85%	64.89%	64.85%
		Mean	66.90%	66.90%	67.75%	64.85%	64.89%	64.85%

TABLE 9. Experiment 1 using FastText (wiki-news-300d-1M.vec and crawl-300d-2M.vec).

Pretrained Model	Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
			Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
wiki-news-300d-1M.vec	Gensim	Sum	66.46%	66.46%	66.77%	61.67%	61.70%	61.67%
		Mean	66.46%	66.46%	66.77%	61.67%	61.70%	61.67%
	SpaCy	Sum	66.90%	66.90%	**67.22%**	61.69%	61.72%	61.69%
		Mean	66.90%	66.90%	**67.22%**	61.69%	61.72%	61.69%
	NTLK	Sum	66.24%	66.25%	67.08%	61.52%	61.55%	61.52%
		Mean	66.24%	66.25%	67.08%	61.52%	61.55%	61.52%
crawl-300d-2M.vec	Gensim	Sum	66.95%	66.96%	67.70%	67.24%	67.24%	67.24%
		Mean	66.95%	66.96%	67.70%	67.24%	67.24%	67.24%
	SpaCy	Sum	67.45%	67.47%	**68.21%**	67.78%	67.78%	67.78%
		Mean	67.45%	67.47%	**68.21%**	67.78%	67.78%	67.78%
	NTLK	Sum	66.56%	66.57%	67.67%	67.61%	67.60%	67.61%
		Mean	66.56%	66.57%	67.67%	67.61%	67.60%	67.61%

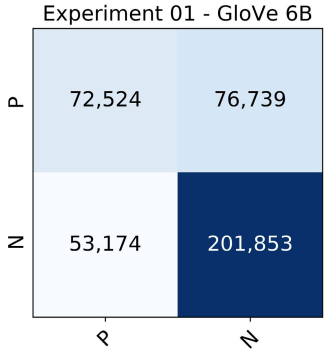

FIGURE 10. Experiment 1 GloVe 6B confusion matrix.

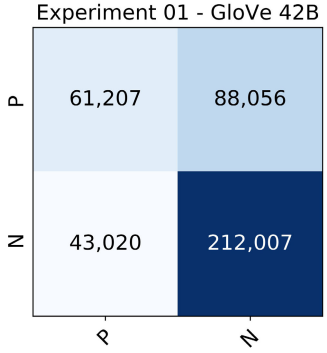

FIGURE 11. Experiment 1 GloVe 42B confusion matrix.

pre-trained model (Table 8) reports that all of the accuracies are in the range of 64% to 69%. The best accuracy (68.09%) is achieved using the SpaCy package with the 90% threshold and Cosine similarity score. Figure 12 shows the confusion matrix of the best score record.

The FastText News pre-trained model (Table 9) reports that all of the accuracies are in the range of 66% to 68%. The best accuracy (67.22%) is achieved using the SpaCy package with the 90% threshold and Cosine similarity score. Figure 13 shows the confusion matrix of the best score record. The FastText Crawl pre-trained model (Table 9) reports that all of the accuracies are in the range of 66% to 69%. The best accuracy (68.21%) is achieved using the SpaCy package with the 90% threshold and Cosine similarity score. Figure 14 shows the confusion matrix of the best score record.

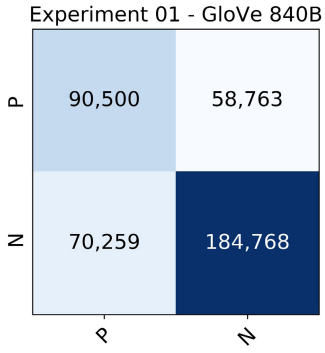

FIGURE 12. Experiment 1 GloVe 840B confusion matrix.

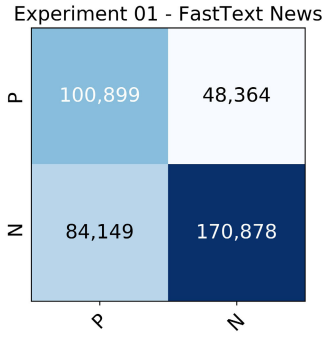

FIGURE 13. Experiment 1 FastText News confusion matrix.

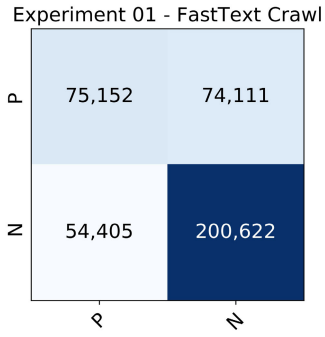

FIGURE 14. Experiment 1 FastText Crawl confusion matrix.

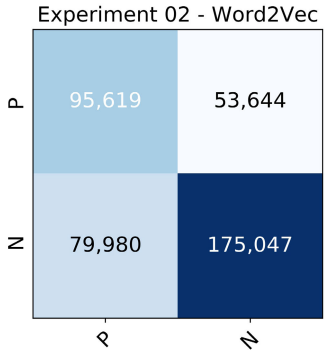

FIGURE 15. Experiment 2 Word2Vec confusion matrix.

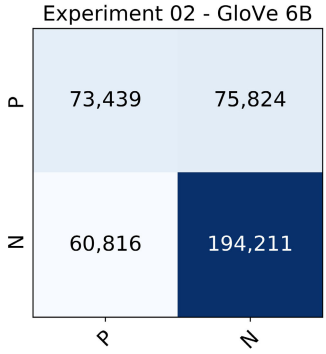

FIGURE 16. Experiment 2 GloVe 6B confusion matrix.

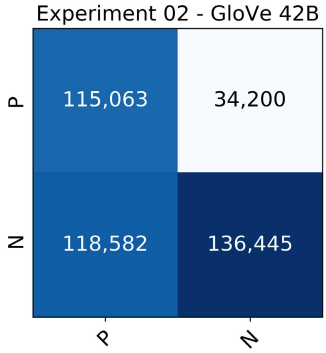

FIGURE 17. Experiment 2 GloVe 42B confusion matrix.

Experiment 2*:* It is similar to Experiment 1 but in the strict mode. The "Is Strict?" flag is set to True in the third layer header. The results are captured in Table 10, Table 11, and Table 12 for Word2Vec, GloVe, and FastText respectively.

From Experiment 2, the GoogleNews Word2Vec pre-trained model (Table 10) reports that all of the accuracies are in the range of 64% to 67%. The best accuracy (66.95%) is achieved using the NLTK package with the 85% threshold and the two scores and with the 90% threshold and Pearson correlation. Figure 15 shows the confusion matrix of the best score record.

The GloVe 6B pre-trained model (Table 11) reports that all of the accuracies are in the range of 50% to 67%. The best accuracy (66.20%) is achieved using the SpaCy package with the 95% threshold and Cosine similarity score. Figure 16 shows the confusion matrix of the best score record. The GloVe 42B pre-trained model (Table 11) reports that all of the accuracies are in the range of 38% to 63%. The best accuracy (62.21%) is achieved using the Gensim package with the 90% threshold and Pearson correlation score. Figure 17 shows the confusion matrix of the best score record. The GloVe 840B pre-trained model (Table 11) reports that all of the accuracies are in the range of 49% to 67%. The best accuracy (66.84%) is achieved using the SpaCy package with the 95% threshold and Cosine similarity score. Figure 18 shows the confusion matrix of the best score record.

TABLE 10. Experiment 2 using Word2Vec (GoogleNews-vectors-negative300.bin).

Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
		Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
Gensim	Sum	64.41%	64.41%	66.51%	66.91%	66.91%	66.91%
	Mean	64.41%	64.41%	66.51%	66.91%	66.91%	66.91%
SpaCy	Sum	64.45%	64.45%	66.57%	66.92%	66.93%	66.92%
	Mean	64.45%	64.45%	66.57%	66.92%	66.93%	66.92%
NTLK	Sum	64.33%	64.33%	66.47%	**66.95%**	**66.95%**	**66.95%**
	Mean	64.33%	64.33%	66.47%	**66.95%**	**66.95%**	**66.95%**

TABLE 11. Experiment 2 using GloVe (glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt).

Pretrained Model	Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
			Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
glove.6B.300d.txt	Gensim	Sum	65.67%	65.67%	62.97%	55.37%	55.40%	55.37%
		Mean	65.67%	65.67%	62.97%	55.37%	55.40%	55.37%
	SpaCy	Sum	**66.20%**	66.20%	60.47%	50.67%	50.72%	50.67%
		Mean	**66.20%**	66.20%	60.47%	50.67%	50.72%	50.67%
	NTLK	Sum	66.10%	66.10%	60.42%	50.74%	50.78%	50.74%
		Mean	66.10%	66.10%	60.42%	50.74%	50.78%	50.74%
glove.42B.300d.txt	**Gensim**	Sum	62.09%	**62.21%**	47.86%	41.29%	41.22%	41.29%
		Mean	62.09%	**62.21%**	47.86%	41.29%	41.22%	41.29%
	SpaCy	Sum	59.86%	60.01%	43.72%	38.92%	38.88%	38.92%
		Mean	59.86%	60.01%	43.72%	38.92%	38.88%	38.92%
	NTLK	Sum	60.09%	60.24%	43.97%	39.04%	39.00%	39.04%
		Mean	60.09%	60.24%	43.97%	39.04%	39.00%	39.04%
glove.840B.300d.txt	Gensim	Sum	66.39%	66.38%	61.30%	52.37%	52.42%	52.37%
		Mean	66.39%	66.38%	61.30%	52.37%	52.42%	52.37%
	SpaCy	Sum	**66.84%**	66.83%	59.75%	49.86%	49.92%	49.86%
		Mean	**66.84%**	66.83%	59.75%	49.86%	49.92%	49.86%
	NTLK	Sum	66.78%	66.78%	60.00%	50.19%	50.25%	50.19%
		Mean	66.78%	66.78%	60.00%	50.19%	50.25%	50.19%

TABLE 12. Experiment 2 using FastText (wiki-news-300d-1M.vec and crawl-300d-2M.vec).

Pretrained Model	Tokenizer Package	Overall Vector	95% Threshold		90% Threshold		85% Threshold	
			Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
wiki-news-300d-1M.vec	Gensim	Sum	64.89%	64.89%	67.23%	65.01%	64.99%	65.01%
		Mean	64.89%	64.89%	67.23%	65.01%	64.99%	65.01%
	SpaCy	Sum	65.28%	65.28%	**67.59%**	63.44%	63.42%	63.44%
		Mean	65.28%	65.28%	**67.59%**	63.44%	63.42%	63.44%
	NTLK	Sum	65.16%	65.16%	67.54%	63.65%	63.63%	63.65%
		Mean	65.16%	65.16%	67.54%	63.65%	63.63%	63.65%
crawl-300d-2M.vec	Gensim	Sum	65.85%	65.84%	57.42%	47.10%	47.13%	47.10%
		Mean	65.85%	65.84%	57.42%	47.10%	47.13%	47.10%
	SpaCy	Sum	**66.19%**	66.18%	54.35%	43.95%	43.97%	43.95%
		Mean	**66.19%**	66.18%	54.35%	43.95%	43.97%	43.95%
	NTLK	Sum	66.16%	66.15%	54.81%	44.25%	44.28%	44.25%
		Mean	66.16%	66.15%	54.81%	44.25%	44.28%	44.25%

The FastText News pre-trained model (Table 12) reports that all of the accuracies are in the range of 63% to 68%. The best accuracy (67.59%) is achieved using the SpaCy package with the 90% threshold and Cosine similarity score. Figure 19 shows the confusion matrix of the best score record. The FastText Crawl pre-trained model (Table 12) reports that all of the accuracies are in the range of 43% to 67%. The best accuracy (66.19%) is achieved using the SpaCy package with the 95% threshold and Cosine similarity score. Figure 20 shows the confusion matrix of the best score record.

Experiment 3: It is similar to Experiment 1, but instead of using the word embedding, the paragraph embedding using USE is used. The "Is Strict?" flag is set to False in the third layer header. The pre-trained models of the USE are USE v4 and v5 (Encoder of greater-than-word length text trained on a variety of data) and USE v3 (Greater-than-word length text encoder for question-answer retrieval). USE v5 is larger than USE v4. The cosine similarity and Pearson correlation are calculated. The 95%, 90%, and 85% thresholds (if above, it is duplicate, and otherwise, it is not duplicate) are set to determine the number of matches. The accuracy is calculated

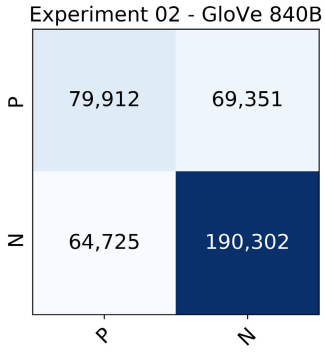

FIGURE 18. Experiment 2 GloVe 840B confusion matrix.

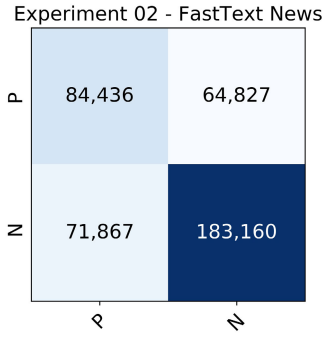

FIGURE 19. Experiment 2 FastText News confusion matrix.

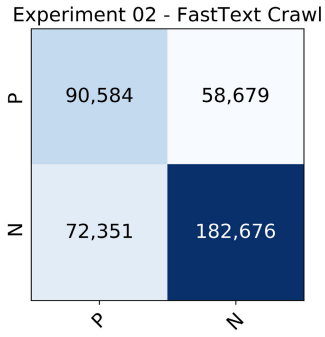

FIGURE 20. Experiment 2 FastText Crawl confusion matrix.

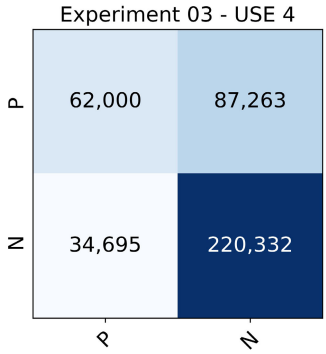

FIGURE 21. Experiment 3 USE v4 confusion matrix.

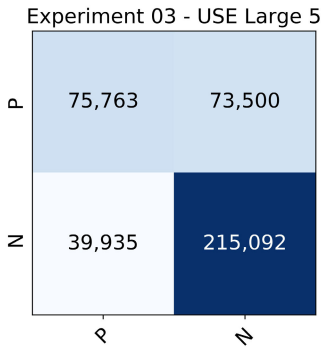

FIGURE 22. Experiment 3 USE Large v5 confusion matrix.

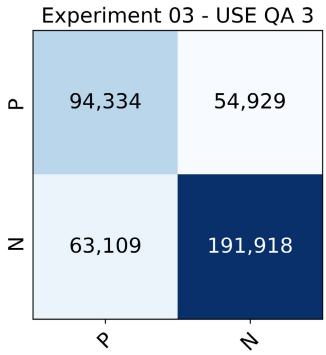

FIGURE 23. Experiment 3 USE QA v3 confusion matrix.

by dividing the number of correct matches by the actual number of records. The results are captured in Table 13.

From Experiment 3 (Table 13), the USE v4 pre-trained model reports that all of the accuracies are in the range of 65% to 70%. The best accuracy (69.83%) is achieved using the SpaCy package with the 85% and 90% thresholds and Pearson correlation score. Figure 21 shows the confusion matrix of the best score record. The USE Large v5 pre-trained model reports that all of the accuracies are in the range of 67% to 72%. The best accuracy (71.94%) is achieved using the SpaCy package with the 85% and the two scores and with the 90% threshold and the Pearson correlation score. Figure 22 shows the confusion matrix of the best score record. The USE QA v3 pre-trained model reports that all of the accuracies are in the range of 67% to 71%. The best accuracy (70.80%) is achieved using the SpaCy package with the 85% and the two scores and with the 90% threshold and the Pearson correlation score. Figure 23 shows the confusion matrix of the best score record.

Experiment 4: It is similar to Experiment 3 but in the strict mode. The "Is Strict?" flag is set to True in the third layer header. The results are captured in Table 14.

From Experiment 4 (Table 14), the USE v4 pre-trained model reports that all of the accuracies are in the range of 65% to 72%. The best accuracy (71.83%) is achieved using the SpaCy package with the 85% and 90% thresholds and Pearson correlation score. Figure 24 shows the confusion

TABLE 13. Experiment 3 using USE v4, Large v5, and QA v3.

Pretrained Model	Tokenizer Package	95% Threshold		90% Threshold		85% Threshold	
		Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
USE v4	Gensim	66.54%	66.54%	67.76%	68.99%	68.98%	68.99%
	SpaCy	67.24%	67.24%	68.54%	**69.83%**	69.82%	**69.83%**
	NTLK	65.64%	65.64%	67.24%	68.77%	68.76%	68.77%
USE Large v5	Gensim	67.95%	67.95%	69.83%	71.06%	71.05%	71.06%
	SpaCy	68.65%	68.66%	70.60%	**71.94%**	**71.94%**	**71.94%**
	NTLK	67.38%	67.38%	69.66%	71.19%	71.19%	71.19%
USE QA v3	Gensim	68.36%	68.36%	69.78%	69.91%	69.90%	69.91%
	SpaCy	69.08%	69.09%	70.53%	**70.80%**	**70.80%**	**70.80%**
	NTLK	67.77%	67.76%	69.62%	70.19%	70.19%	70.19%

TABLE 14. Experiment 4 using USE v4, Large v5, and QA v3.

Pretrained Model	Tokenizer Package	95% Threshold		90% Threshold		85% Threshold	
		Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
USE v4	Gensim	65.25%	65.25%	68.80%	71.58%	71.57%	71.58%
	SpaCy	65.49%	65.49%	69.05%	**71.83%**	71.82%	**71.83%**
	NTLK	65.40%	65.40%	68.96%	71.77%	71.76%	71.77%
USE Large v5	Gensim	69.98%	69.98%	75.16%	77.55%	77.56%	77.55%
	SpaCy	70.25%	70.26%	75.52%	**77.95%**	**77.95%**	**77.95%**
	NTLK	70.10%	70.10%	75.35%	77.85%	77.86%	77.85%
USE QA v3	Gensim	69.70%	69.70%	74.78%	77.16%	77.17%	77.16%
	SpaCy	69.97%	69.98%	75.12%	**77.54%**	**77.54%**	**77.54%**
	NTLK	69.83%	69.83%	74.97%	77.44%	77.44%	77.44%

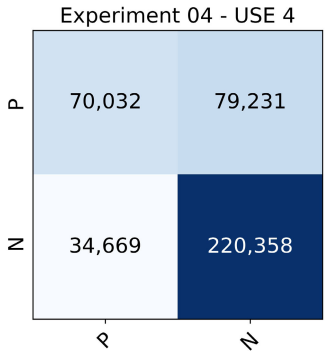

FIGURE 24. Experiment 4 USE v4 confusion matrix.

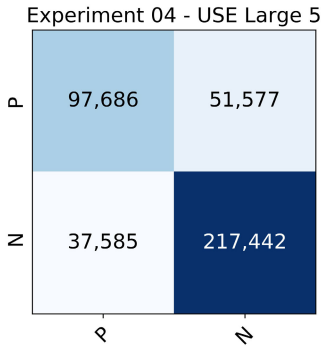

FIGURE 25. Experiment 4 USE Large v5 confusion matrix.

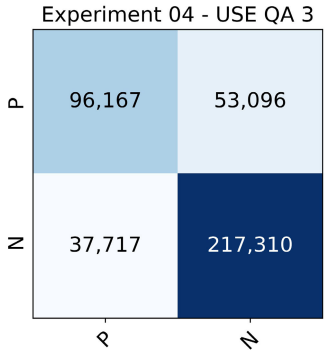

FIGURE 26. Experiment 4 USE QA v3 confusion matrix.

matrix of the best score record. The USE Large v5 pre-trained model reports that all of the accuracies are in the range of 69% to 78%. The best accuracy (77.95%) is achieved using the SpaCy package with the 85% and the two scores and with the 90% threshold and Pearson correlation score. Figure 25 shows the confusion matrix of the best score record. The USE QA v3 pre-trained model reports that all of the accuracies are in the range of 69% to 78%. The best accuracy (77.54%) is achieved using the SpaCy package with the 90% threshold and Pearson correlation score and with the 85% threshold with the two scores. Figure 26 shows the confusion matrix of the best score record.

Table 15 summarizes the results achieved by Experiments 1 to 4 with the corresponding thresholds, scores, and packages.

Table 15 shows that the best-reported accuracy (77.95%) is achieved by the USE Larges v5 in the "Strict" mode using the SpaCy package. SpaCy package reports the best accuracies in most of the records (15 out of 18) over Gensim and NLTK packages. The threshold (90%) is reported as the

TABLE 15. Experiments 1 to 4 summarization.

Experiment	Model	Restrict	Best Accuracy	Package	Threshold	Scores
1	Word2Vec	FALSE	67.11%	NLTK	85%	Cosine
	GloVe 6B	FALSE	67.87%	SpaCy	90%	Cosine
	GloVe 42B	FALSE	67.58%	SpaCy	95%	Cosine
	GloVe 840B	FALSE	68.09%	SpaCy	90%	Cosine
	FastText News	FALSE	67.22%	SpaCy	90%	Cosine
	FastText Crawl	**FALSE**	**68.21%**	**SpaCy**	**90%**	**Cosine**
2	Word2Vec	TRUE	66.95%	NLTK	90% and 85%	Cosine and Pearson
	GloVe 6B	TRUE	66.20%	SpaCy	95%	Cosine
	GloVe 42B	TRUE	66.21%	Gensim	95%	Pearson
	GloVe 840B	TRUE	66.84%	SpaCy	95%	Cosine
	FastText News	**TRUE**	**67.59%**	**SpaCy**	**90%**	**Cosine**
	FastText Crawl	TRUE	66.19%	SpaCy	95%	Cosine
3	USE v4	FALSE	69.83%	SpaCy	90% and 85%	Pearson
	USE Large v5	**FALSE**	**71.94%**	**SpaCy**	**90% and 85%**	**Cosine and Pearson**
	USE QA v3	FALSE	70.80%	SpaCy	90% and 85%	Cosine and Pearson
4	USE v4	TRUE	71.83%	SpaCy	90% and 85%	Pearson
	USE Large v5	**TRUE**	**77.95%**	**SpaCy**	**90% and 85%**	**Cosine and Pearson**
	USE QA v3	TRUE	77.54%	SpaCy	90% and 85%	Cosine and Pearson

TABLE 16. Experiment 5 using Word2Vec (GoogleNews-vectors-negative300.bin).

Tokenizer Package	Overall Vector	Cosine	Pearson
Gensim	Sum	1.73	1.73
	Mean	1.73	1.73
SpaCy	Sum	1.73	1.73
	Mean	1.73	1.73
NTLK	Sum	**1.69**	1.70
	Mean	**1.69**	1.70

most suitable one (12 out of 18). The Cosine similarity score is better than the Pearson correlation score (15 out of 18).

Experiment 5*:* Similar to the previous experiments, in the pre-processing phase, whitespace, punctuations, stop-words, and accents are removed and lemmatization is applied (the "Is Strict?" flag is set to False in the third layer header). The tokenization phase is applied and the text is converted into words. The overall text vector is calculated using the sum and mean (average) of the word vectors. The pre-trained models of Word2Vec (i.e. GoogleNews-vectors-negative300.bin), GloVe (i.e. glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt), and FastText (i.e. wiki-news-300d-1M.vec and crawl-300d-2M.vec) are used in the word embedding phase. The cosine similarity and Pearson correlation are calculated and based on them, the corresponding predicted grade is calculated by dividing the score by 100 and multiplying it with 5. The root mean square error (RMSE) between the predicted scores ($\hat{y}_i$) and actual dataset average scores (y_i) is calculated in Equation 6) where *Ns* is the number of elements. The RMSE results are captured in Table 16, Table 17, and Table 18 for Word2Vec, GloVe, and FastText respectively.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{Ns}(\hat{y}_i - y_i)^2}{Ns}} \tag{6}$$

From Experiment 5, the GoogleNews Word2Vec pre-trained model (Table 16) reports the lowest RMSE value of 1.69. It is achieved using the NLTK package and Cosine similarity score. The overall vector calculation using the sum and mean reported the same results which indicate that the cosine similarity and Pearson correlation are not affected by that. The GloVe pre-trained model (Table 17) reports the lowest RMSE value of 1.49. It is achieved using the NLTK package with the two scores. The GloVe 42B pre-trained model (Table 17) reports the lowest RMSE value of 1.17. It is achieved using the Gensim and NLTK packages with the two scores. The GloVe 840B pre-trained model (Table 17) reports the lowest RMSE value of 1.19. It is achieved using the NLTK package with the two scores. The FastText News pre-trained model (Table 18) reports the lowest RMSE value of 1.09. It is achieved using the Gensim and NLTK packages with the two scores. The FastText Crawl pre-trained model (Table 18) reports the lowest RMSE value of 1.27. It is achieved using the NLTK package with the two scores.

Experiment 6*:* It is similar to Experiment 5 but in the strict mode. The "Is Strict?" flag is set to True in the third layer header. The results are captured in Table 19, Table 20, and Table 21 for Word2Vec, GloVe, and FastText respectively.

From Experiment 6, the GoogleNews Word2Vec pre-trained model (Table 19) reports the lowest RMSE value of 1.43. It is achieved using the Gensim package and Cosine similarity score. The GloVe pre-trained model (Table 20) reports the lowest RMSE value of 1.17. It is achieved using the NLTK and SpaCy packages with the two scores. The GloVe 42B pre-trained model (Table 20) reports the lowest RMSE value of 1.18. It is achieved using the Gensim package with the two scores. The GloVe 840B pre-trained model (Table 20) reports the lowest RMSE value of 1.14. It is achieved using the Gensim package with two scores. The FastText News pre-trained model (Table 21) reports the lowest RMSE value of 1.10. It is achieved using the Gensim package with the two scores. The FastText Crawl pre-trained

TABLE 17. Experiment 5 using GloVe (glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt).

Pretrained Model	Tokenizer Package	Overall Vector	Cosine	Pearson
glove.6B.300d.txt	Gensim	Sum	1.58	1.58
		Mean	1.58	1.58
	SpaCy	Sum	1.54	1.54
		Mean	1.54	1.54
	NTLK	Sum	**1.49**	**1.49**
		Mean	**1.49**	**1.49**
glove.42B.300d.txt	**Gensim**	Sum	**1.17**	**1.17**
		Mean	**1.17**	**1.17**
	SpaCy	Sum	1.19	1.19
		Mean	1.19	1.19
	NTLK	Sum	**1.17**	**1.17**
		Mean	**1.17**	**1.17**
glove.840B.300d.txt	Gensim	Sum	1.24	1.24
		Mean	1.24	1.24
	SpaCy	Sum	1.22	1.22
		Mean	1.22	1.22
	NTLK	Sum	**1.19**	**1.19**
		Mean	**1.19**	**1.19**

TABLE 18. Experiment 5 using FastText (wiki-news-300d-1M.vec and crawl-300d-2M.vec).

Pretrained Model	Tokenizer Package	Overall Vector	Cosine	Pearson
wiki-news-300d-1M.vec	**Gensim**	Sum	**1.09**	**1.09**
		Mean	**1.09**	**1.09**
	SpaCy	Sum	1.12	1.12
		Mean	1.12	1.12
	NTLK	Sum	**1.09**	**1.09**
		Mean	**1.09**	**1.09**
crawl-300d-2M.vec	Gensim	Sum	1.31	1.31
		Mean	1.31	1.31
	SpaCy	Sum	1.30	1.30
		Mean	1.30	1.30
	NTLK	Sum	**1.27**	**1.27**
		Mean	**1.27**	**1.27**

TABLE 19. Experiment 6 using Word2Vec (GoogleNews-vectors-negative300.bin).

Tokenizer Package	Overall Vector	Cosine	Pearson
Gensim	Sum	**1.43**	1.44
	Mean	**1.43**	1.44
SpaCy	Sum	1.47	1.47
	Mean	1.47	1.47
NTLK	Sum	1.47	1.47
	Mean	1.47	1.47

model (Table 21) reports the lowest RMSE value of 1.09. It is achieved using the NLTK package with the Pearson correlation score.

Experiment 7: It is similar to Experiment 6, but instead of using the word embedding, the paragraph embedding using USE is used. The "Is Strict?" flag is set to False in the third layer header. The pre-trained models of the USE are USE v4, USE Large v5, and USE QA v3. The results are captured in Table 22.

From Experiment 7 (Table 22), the USE v4 pre-trained model reports the lowest RMSE value of 7.13. It is achieved using the SpaCy package with the two scores. The USE Large v5 pre-trained model reports the lowest RMSE value of 7.24. It is achieved using the SpaCy package with the two scores. The USE QA v3 pre-trained model reports the lowest RMSE value of 7.38. It is achieved using the SpaCy package with the two scores.

Experiment 8: It is similar to Experiment 7 but in the strict mode. The "Is Strict?" flag is set to True in the third layer header. The results are captured in Table 23.

From Experiment 8 (Table 23), the USE v4 pre-trained model reports the lowest RMSE value of 7.11. It is achieved using the SpaCy package with the two scores. The USE Large v5 pre-trained model reports the lowest RMSE value of 7.15. It is achieved using the SpaCy package with the two scores. The USE QA v3 pre-trained model reports the lowest RMSE value of 7.41. It is achieved using the SpaCy package with the two scores.

TABLE 20. Experiment 6 using GloVe (glove.6B.300d.txt, glove.42B.300d.txt, and glove.840B.300d.txt).

Pretrained Model	Tokenizer Package	Overall Vector	Cosine	Pearson
glove.6B.300d.txt	Gensim	Sum	1.20	1.20
		Mean	1.20	1.20
	SpaCy	Sum	**1.17**	**1.17**
		Mean	**1.17**	**1.17**
	NTLK	Sum	**1.17**	**1.17**
		Mean	**1.17**	**1.17**
glove.42B.300d.txt	**Gensim**	Sum	**1.18**	**1.18**
		Mean	**1.18**	**1.18**
	SpaCy	Sum	1.21	1.20
		Mean	1.21	1.20
	NTLK	Sum	1.21	1.21
		Mean	1.21	1.21
glove.840B.300d.txt	**Gensim**	Sum	**1.14**	**1.14**
		Mean	**1.14**	**1.14**
	SpaCy	Sum	1.16	1.16
		Mean	1.16	1.16
	NTLK	Sum	1.16	1.16
		Mean	1.16	1.16

TABLE 21. Experiment 6 using FastText (wiki-news-300d-1M.vec and crawl-300d-2M.vec).

Pretrained Model	Tokenizer Package	Overall Vector	Cosine	Pearson
wiki-news-300d-1M.vec	**Gensim**	Sum	**1.10**	**1.10**
		Mean	**1.10**	**1.10**
	SpaCy	Sum	1.13	1.13
		Mean	1.13	1.13
	NTLK	Sum	1.12	1.12
		Mean	1.12	1.12
crawl-300d-2M.vec	Gensim	Sum	1.11	1.11
		Mean	1.11	1.11
	SpaCy	Sum	1.10	1.10
		Mean	1.10	1.10
	NTLK	Sum	1.10	**1.09**
		Mean	1.10	**1.09**

TABLE 22. Experiment 7 using USE v4, Large v5, and QA v3.

Pretrained Model	Tokenizer Package	Cosine	Pearson
USE v4	Gensim	7.15	7.15
	SpaCy	**7.13**	**7.13**
	NTLK	7.16	7.16
USE Large v5	Gensim	7.26	7.27
	SpaCy	**7.24**	**7.24**
	NTLK	7.39	7.39
USE QA v3	Gensim	7.42	7.42
	SpaCy	**7.38**	**7.38**
	NTLK	7.52	7.52

TABLE 23. Experiment 8 using USE v4, Large v5, and QA v3.

Pretrained Model	Tokenizer Package	Cosine	Pearson
USE v4	Gensim	7.16	7.16
	SpaCy	**7.11**	**7.11**
	NTLK	7.13	7.13
USE Large v5	Gensim	7.19	7.19
	SpaCy	**7.15**	**7.15**
	NTLK	7.18	7.18
USE QA v3	Gensim	7.43	7.43
	SpaCy	**7.41**	**7.41**
	NTLK	7.44	7.44

Table 24 summarizes the results achieved by Experiments 5 to 8 with the corresponding scores and packages.

Table 24 shows that the best-reported lowest RMSE (1.09) is achieved by the FastText News in the non-"Strict" mode using the NLTK and Gensim packages and by the FastText Crawl in the "Strict" mode using the SpaCy package. The three Python packages are suitable as they report the same values in most of the records. The two scores are suitable as they report the same values in most of the records.

B. EXPRESSIONS MATCHING EXPERIMENTS

The expression matching experiments are performed on a dataset (named HMB-EMD-v1) compiled by the authors. It consists of 5 columns (Number, Expression 1, Expression 2, Are Equal?, Variables). The first column is a counter, the second two columns are two expressions, the third column is to indicate if they are equal or not, and the last column lists the variables used in the expressions. The dataset consists of 150 records. Table 25 shows 5 random records of the dataset.

TABLE 24. Experiments 5 to 8 summarization.

Experiment	Model	Restrict	Lowerest RMSE	Package	Scores
5	Word2Vec	FALSE	1.69	NLTK	Cosine
	GloVe 6B	FALSE	1.49	NLTK	Cosine and Pearson
	GloVe 42B	FALSE	1.17	NLTK and Gensim	Cosine and Pearson
	GloVe 840B	FALSE	1.19	NLTK	Cosine and Pearson
	FastText News	**FALSE**	**1.09**	**NLTK and Gensim**	**Cosine and Pearson**
	FastText Crawl	FALSE	1.27	NLTK	Cosine and Pearson
6	Word2Vec	TRUE	1.43	Gensim	Cosine and Pearson
	GloVe 6B	TRUE	1.17	SpaCy and NLTK	Cosine and Pearson
	GloVe 42B	TRUE	1.18	Gensim	Cosine and Pearson
	GloVe 840B	TRUE	1.14	Gensim	Cosine and Pearson
	FastText News	TRUE	1.10	Gensim	Cosine and Pearson
	FastText Crawl	**TRUE**	**1.09**	**SpaCy**	**Pearson**
7	**USE v4**	**FALSE**	**7.13**	**SpaCy**	**Cosine and Pearson**
	USE Large v5	FALSE	7.24	SpaCy	Cosine and Pearson
	USE QA v3	FALSE	7.38	SpaCy	Cosine and Pearson
8	**USE v4**	**TRUE**	**7.11**	**SpaCy**	**Cosine and Pearson**
	USE Large v5	TRUE	7.15	SpaCy	Cosine and Pearson
	USE QA v3	TRUE	7.41	SpaCy	Cosine and Pearson

TABLE 25. Five random rows of the HMB-EMD-v1 dataset.

Number	Expression 1	Expression 2	Are Equal?	Variables
25	(5*x*y/y+x*0)*2*x*y*z	5*x*2*x*y*z	1	x,y,z
50	(a*b)/c*(x*y)+z*0	a*b/c*x*y	1	a,b,c,x,y,z
75	1+5*5+5*0	1+5*5	1	
100	a*x+2*z*y+1*0	a*x+2*z*y	1	a,x,y,z
125	(3*a*b*(a+2))*d	3*a*b*a+3*a*b*2	0	a,b,d

TABLE 26. The achieved results from experiment 9.

Approach	Number of Matches	Accuracy
Proposed Algorithm	150	100%
SymPy Python Package	107	71.33%

Experiment 9: The suggested algorithm is applied to the proposed dataset and the number of matches is reported. The SymPy Python package is applied also to the dataset. Table 26 reports the achieved results.

Table 26 shows that the suggested algorithm (HMB-MMS-EMA) achieved an accuracy of 100% which is more than the Sympy Python package on the 150 records of the proposed dataset.

TABLE 27. Table of Abbreviations.

Abbreviation	Description
AECF	Automatic Exam Correction Framework
BERT	Bidirectional Encoder Representations from Transformers
CAS	Computer Algebra System
CBOW	Common Bag of Words
DAN	Deep Averaging Network
GloVe	Global Vectors
GPT-3	Generative Pre-trained Transformer 3
IDF	Inverse Document Frequency
MCQ	Multiple Choice Question
MOOC	Massive Open Online Course
NLP	Natural Language Processing
NSP	Next Sentence Prediction
PVDM	Distributed Memory version of Paragraph Vector
PVDOBM	Distributed Bag of Words version of Paragraph Vector
RMSE	Root Mean Square Error
RoBERTa	Robustly optimized BERT approach
SNLI	Stanford Natural Language Inference
TF	Term Frequency
TF-IDF	Term Frequency-Inverse Document Frequency
TPU	Tensor Processing Unit
USE	Universal Sentence Encoder

V. CONCLUSION AND FUTURE WORK

The current work reviewed the literature on text semantic similarity and automatic exam correction systems. It proposed an automatic exam correction framework (HMB-AECF) for MCQs, essays, and equations that was abstracted into five layers. Each layer had its separate work and header (unless the last one). It compared the different approaches to convert (i.e. embed) the textual data such as essays and short answers into numerical data. The Word2Vec, FastText, Glove, and Universal Sentence Encoder (USE) were used as the embedding pre-trained models in the experiments while BERT, SentenceBERT, RoBERTa, XLNET, and GPT-series were mentioned and can be depended on in future studies. The comparison was performed using three well-known Python packages (Gensim, SpaCy, and NLTK) in eight experiments. The experiments were performed on the Quora Questions Pairs and the UNT Computer Science Short Answer datasets. The best-achieved highest accuracy in the first four experiments was 77.95% without fine-tuning the pre-trained models by the USE. The best-achieved lowest RMSE in the second four experiments was 1.09 without fine-tuning the used pre-trained models by the USE. The current study also proposed an equations' similarity checker algorithm named HMB-MMS-EMA. It presented an expression matching dataset named HMB-EMD-v1. The ninth experiment was performed as a comparison between the HMB-MMS-EMA and the SymPy Python package. The HMB-MMS-EMA reported 100% accuracy over the SymPy Python package

which reported 71.33% only. As the future work, the HMB-AECF can be extended for other types of questions and the HMB-MMS-EMA algorithm can be improved to carry other arithmetic operations and functions such as sine and cosine. Other machine learning and state-of-the-art deep learning techniques, approaches, and packages can be used for the paper evaluations.

APPENDICES

TABLE OF ABBREVIATIONS

Table 27 presents the ''Table of Abbreviations'' and is ordered in ascending order.

TABLE OF SYMBOLS

Table 28 presents the ''Table of Symbols'' and is ordered according to the symbols' existence.

TABLE 28. Table of symbols.

Symbol	Description
t	The term
d	The current document
D	The documents in the corpus
N	The number of total documents in the corpus
n	The number of elements
A, B	Numerical vectors
W	The left branch elements count
M	The right branch elements count
N_s	The number of elements in the RMSE equation
i	A counter (i.e. iterator)
y_i	The actual dataset average scores
$\hat{y}_i$	The predicted scores

REFERENCES

[1] R. Mihalcea, C. Corley, and C. Strapparava, ''Corpus-based and knowledge-based measures of text semantic similarity,'' in *Proc. AAAI*, vol. 6, 2006, pp. 775–780.
[2] L. Zhiqiang, S. Werimin, and Y. Zhenhua, ''Measuring semantic similarity between words using wikipedia,'' in *Proc. Int. Conf. Web Inf. Syst. Mining*, Nov. 2009, pp. 251–255.
[3] E. T. Al-Shammari, ''Lemmatizing, stemming, and Query expansion method and system,'' U.S. Patent 8 473 279, Jun. 25, 2013.
[4] V. Balakrishnan and E. Lloyd-Yemoh, ''Stemming and lemmatization: A comparison of retrieval performances,'' in *Proc. SCEI Seoul Conf.*, Seoul, South Korea, Apr. 2014. [Online]. Available: http://eprints.um.edu.my/13423/
[5] N. Habash, O. Rambow, and R. Roth, ''MADA+ TOKAN: A toolkit for Arabic tokenization, diacritization, morphological disambiguation, POS tagging, stemming and lemmatization,'' in *Proc. 2nd Int. Conf. Arabic Lang. Resour. Tools (MEDAR)*, Cairo, Egypt, vol. 41, 2009, p. 62.
[6] I. Boban, A. Doko, and S. Gotovac, ''Sentence retrieval using stemming and lemmatization with different length of the queries,'' *Adv. Sci., Technol. Eng. Syst. J.*, vol. 5, no. 3, pp. 349–354, 2020.
[7] R. M. Kaplan, ''Method and apparatus for tokenizing text,'' U.S. Patent 5 721 939, Feb. 24, 1998.
[8] P. McNamee and J. Mayfield, ''Character N-gram tokenization for European language text retrieval,'' *Inf. Retr.*, vol. 7, nos. 1–2, pp. 73–97, Jan. 2004.
[9] G. Carenini, R. T. Ng, and E. Zwart, ''Extracting knowledge from evaluative text,'' in *Proc. 3rd Int. Conf. Knowl. Capture*, 2005, pp. 11–18.
[10] N. Azam and J. Yao, ''Comparison of term frequency and document frequency based feature selection metrics in text categorization,'' *Expert Syst. Appl.*, vol. 39, no. 5, pp. 4760–4768, Apr. 2012.
[11] A. M. Dai, C. Olah, and Q. V. Le, ''Document embedding with paragraph vectors,'' 2015, *arXiv:1507.07998*. [Online]. Available: http://arxiv.org/abs/1507.07998
[12] M. Mohd, R. Jan, and M. Shah, ''Text document summarization using word embedding,'' *Expert Syst. Appl.*, vol. 143, Apr. 2020, Art. no. 112958.
[13] A. Kulkarni and A. Shivananda, ''Converting text to features,'' in *Natural Language Processing Recipes*. Berkeley, CA, USA: Apress, 2019, doi: 10.1007/978-1-4842-4267-4_3.
[14] H. Christian, M. P. Agus, and D. Suhartono, ''Single document automatic text summarization using term frequency-inverse document frequency (TF-IDF),'' *ComTech, Comput., Math. Eng. Appl.*, vol. 7, no. 4, pp. 285–294, 2016.
[15] A. A. Hakim, A. Erwin, K. I. Eng, M. Galinium, and W. Muliady, ''Automated document classification for news article in Bahasa Indonesia based on term frequency inverse document frequency (TF-IDF) approach,'' in *Proc. 6th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Oct. 2014, pp. 1–4.
[16] M. Aydoğan and A. Karci, ''Improving the accuracy using pre-trained word embeddings on deep neural networks for turkish text classification,'' *Phys. A, Stat. Mech. Appl.*, vol. 541, p. 123288, 2020.
[17] M. Grohe, ''Word2vec, node2vec, graph2vec, x2vec: Towards a theory of vector embeddings of structured data,'' in *Proc. 39th ACM SIGMOD-SIGACT-SIGAI Symp. Princ. Database Syst.*, Jun. 2020, pp. 1–16.
[18] J. Pennington, R. Socher, and C. Manning, ''Glove: Global vectors for word representation,'' in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162
[19] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, ''Advances in pre-training distributed word representations,'' in *Proc. Int. Conf. Lang. Resour. Eval.*, 2018, pp. 52–55. [Online]. Available: https://www.aclweb.org/anthology/L18-1.pdf
[20] S. Yilmaz and S. Toklu, ''A deep learning analysis on question classification task using word2vec representations,'' *Neural Comput. Appl.*, vol. 7, pp. 1–20, Jan. 2020.
[21] M. Mohammed and N. Omar, ''Question classification based on Bloom's taxonomy cognitive domain using modified TF-IDF and word2vec,'' *PLoS ONE*, vol. 15, no. 3, Mar. 2020, Art. no. e0230442.
[22] H. Yousuf and S. Salloum, ''Survey analysis: Enhancing the security of vectorization by using word2vec and CryptDB,'' *Adv. Sci., Technol. Eng. Syst. J.*, vol. 5, no. 4, pp. 374–380, 2020.
[23] T. Hai Nguyen, ''Analyze the effects of weighting functions on cost function in the glove model,'' 2020, *arXiv:2009.04732*. [Online]. Available: http://arxiv.org/abs/2009.04732
[24] B. Mansurov and A. Mansurov, ''Development of word embeddings for uzbek language,'' 2020, *arXiv:2009.14384*. [Online]. Available: http://arxiv.org/abs/2009.14384
[25] V. Gaikwad and Y. Haribhakta, ''Adaptive GloVe and FastText model for hindi word embeddings,'' in *Proc. 7th ACM IKDD CoDS 25th COMAD*, Jan. 2020, pp. 175–179.
[26] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, ''Xlnet: Generalized autoregressive pretraining for language understanding,'' in *Proc. Adv. Neural Inf. Process. Syst.*, 2019, pp. 5753–5763.
[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, ''RoBERTa: A robustly optimized BERT pretraining approach,'' 2019, *arXiv:1907.11692*. [Online]. Available: http://arxiv.org/abs/1907.11692
[28] D. Shirafuji, H. Kameya, R. Rzepka, and K. Araki, ''Summarizing utterances from japanese assembly minutes using political sentence-BERT-based method for QA Lab-PoliInfo-2 task of NTCIR-15,'' 2020, *arXiv:2010.12077*. [Online]. Available: http://arxiv.org/abs/2010.12077
[29] N. Reimers and I. Gurevych, ''Sentence-BERT: Sentence embeddings using Siamese BERT-networks,'' 2019, *arXiv:1908.10084*. [Online]. Available: https://arxiv.org/abs/1908.10084
[30] H. A. M. Hassan, G. Sansonetti, F. Gasparetti, A. Micarelli, and J. Beel, ''Bert, elmo, use and infersent sentence encoders: The panacea for research-paper recommendation?'' in *Proc. RecSys*, 2019, pp. 6–10.
[31] Y. Yang, D. Cer, A. Ahmad, M. Guo, J. Law, N. Constant, G. Hernandez Abrego, S. Yuan, C. Tar, Y.-H. Sung, B. Strope, and R. Kurzweil, ''Multilingual universal sentence encoder for semantic retrieval,'' 2019, *arXiv:1907.04307*. [Online]. Available: http://arxiv.org/abs/1907.04307
[32] M.-Y. Day and C. Jou, ''Universal sentence-embedding models,'' *Foundation*, vol. 2, nos. 03–2020, p. 09, 2020.
[33] T. B. Brown *et al.*, ''Language models are few-shot learners,'' 2020, *arXiv:2005.14165*. [Online]. Available: http://arxiv.org/abs/2005.14165
[34] H. T. Nguyen, P. H. Duong, and E. Cambria, ''Learning short-text semantic similarity with word embeddings and external knowledge sources,'' *Knowl.-Based Syst.*, vol. 182, Oct. 2019, Art. no. 104842.

[35] O. Araque, G. Zhu, and C. A. Iglesias, "A semantic similarity-based perspective of affect lexicons for sentiment analysis," *Knowl.-Based Syst.*, vol. 165, pp. 346–359, Feb. 2019.
[36] T. Thongtan and T. Phienthrakul, "Sentiment classification using document embeddings trained with cosine similarity," in *Proc. 57th Annu. Meeting Assoc. Comput. Linguistics, Student Res. Workshop*, 2019, pp. 407–414.
[37] S. Bag, S. K. Kumar, and M. K. Tiwari, "An efficient recommendation generation using relevant jaccard similarity," *Inf. Sci.*, vol. 483, pp. 53–64, May 2019.
[38] P. Tabaghi, I. Dokmanić, and M. Vetterli, "Kinetic Euclidean distance matrices," *IEEE Trans. Signal Process.*, vol. 68, pp. 452–465, 2020, doi: 10.1109/TSP.2019.2959260.
[39] Y. Huang, W. Jin, B. Li, P. Ge, and Y. Wu, "Automatic modulation recognition of radar signals based on manhattan distance-based features," *IEEE Access*, vol. 7, pp. 41193–41204, 2019.
[40] M. M. Haider, M. A. Hossin, H. R. Mahi, and H. Arif, "Automatic text summarization using gensim Word2 Vec and K-means clustering algorithm," in *Proc. IEEE Region Symp. (TENSYMP)*, Dec. 2020, pp. 283–286.
[41] B. Srinivasa-Desikan, *Natural Language Processing and Computational Linguistics: A Practical Guide to Text Analysis with Python, Gensim, spaCy, and Keras*. Birmingham, U.K.: Packt, 2018.
[42] J. Perkins, *Python text Processing with NLTK 2.0 Cookbook*. Birmingham, U.K.: Packt, 2010.
[43] R. Rehurek and P. Sojka. (2011). *Gensim-Statistical Semantics in Python*. EuroScipy, Paris, France. [Online]. Available: https://www.fi.muni.cz/usr/sojka/posters/rehurek-sojka-scipy2011.pdf
[44] (2017). *Espacy-Industrial-Strength Natural Language Processing in Python*. [Online]. Available: https://spacy.io
[45] N. Hardeniya, J. Perkins, D. Chopra, N. Joshi, and I. Mathur, *Natural Language Processing: Python and NLTK*. Birmingham, U.K.: Packt, 2016.
[46] K. A. Bollen, "Structural equation models," *Encyclopedia Biostatist.*, vol. 7, p. 51, Jul. 2005.
[47] N. Ramsey, "Unparsing expressions with prefix and postfix operators," *Software: Pract. Exper.*, vol. 28, no. 12, pp. 1327–1356, Oct. 1998.
[48] G. Araujo, P. Centoducatte, R. Azevedo, and R. Pannain, "Expression-tree-based algorithms for code compression on embedded RISC architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 8, no. 5, pp. 530–533, Oct. 2000.
[49] L. Wang, Y. Wang, D. Cai, D. Zhang, and X. Liu, "Translating a math word problem to an expression tree," 2018, *arXiv:1811.05632*. [Online]. Available: http://arxiv.org/abs/1811.05632
[50] M. P. Barnett and X. Rui, "Infix to prefix conversion as a PST reduction," *ACM SIGPLAN Notices*, vol. 25, no. 5, pp. 34–38, 1990.
[51] K. Thompson, "Programming techniques: Regular expression search algorithm," *Commun. ACM*, vol. 11, no. 6, pp. 419–422, Jun. 1968.
[52] D. A. Turner, "Recursion equations as a programming language," in *A List of Successes That Can Change the World* (Lecture Notes in Computer Science), vol. 9600, S. Lindley, C. McBride, P. Trinder, and D. Sannella, Eds. Cham, Switzerland: Springer, 1982, doi: 10.1007/978-3-319-30936-1_24.
[53] A. Meurer, C. P. Smith, M. Paprocki, and O. Certík, "SymPy: Symbolic computing in Python," *PeerJ Comput. Sci.*, vol. 3, p. e103, Jan. 2017.
[54] F. S. Pribadi, T. B. Adji, A. E. Permanasari, A. Mulwinda, and A. B. Utomo, "Automatic short answer scoring using words overlapping methods," *AIP Conf. Proc.*, vol. 1818, Mar. 2017, Art. no. 020042.
[55] N. Sázen, A. N. Gorban, J. Levesley, and E. M. Mirkes, "Automatic short answer grading and feedback using text mining methods," *Procedia Comput. Sci.*, vol. 169, pp. 726–743, Dec. 2020.
[56] M. Mohler, R. Bunescu, and R. Mihalcea, "Learning to grade short answer questions using semantic similarity measures and dependency graph alignments," in *Proc. 49th Annu. Meeting Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2011, pp. 752–762.
[57] S. Hassan, A. A., and M. El-Ramly, "Automatic short answer scoring based on paragraph embeddings," *Int. J. Adv. Comput. Sci. Appl.*, vol. 9, no. 10, pp. 397–402, 2018.
[58] M. Mohler and R. Mihalcea, "Text-to-text semantic similarity for automatic short answer grading," in *Proc. 12th Conf. Eur. Chapter Assoc. Comput. Linguistics*, 2009, pp. 567–575.
[59] Z. Chen, H. Zhang, X. Zhang, and L. Zhao, "Quora question Pairs," Univ. Waterloo, Waterloo, ON, Canada, Tech. Rep., 2018. [Online]. Available: http://static.hongbozhang.me/doc/STAT_441_Report.pdf
[60] A. Prabhudesai and T. N. B. Duong, "Automatic short answer grading using siamese bidirectional LSTM based regression," in *Proc. IEEE Int. Conf. Eng., Technol. Educ. (TALE)*, Dec. 2019, pp. 1–6.

HOSSAM MAGDY BALAHA was born in Egypt in 1993. He received the B.Sc. and M.Sc. degrees from the Computers and Systems Engineering Department, Faculty of Engineering, Mansoura University, Egypt. He is currently an Assistant Lecturer with the Computers and Systems Engineering Department, Faculty of Engineering, Mansoura University, a Senior Full Stack Laravel Web Developer, and Freelancer. His major research interests are web development, the Internet of Things (IoT), deep learning (DL), computer vision (CV), soft computing, embedded systems, and robotics. He made different courses on YouTube including web development and the IoT. He built different systems including web and mobile applications. He had a part in different projects and competitions related to his interests. He has served as a Reviewer in different journals such as the IEEE ACCESS, IEEE TRANSACTIONS ON CYBERNETICS, *Artificial Intelligence Review* (AIRE), and *International Journal on Intelligent Systems* (INT2).

MAHMOUD M. SAAFAN received the B.Sc. degree from the Electronics Engineering Department, and the M.Sc. and Ph.D. degrees in computers and control systems engineering from Mansoura University, Egypt. He is currently an Assistant Professor with the Computers and Control Systems Engineering Department, Faculty of Engineering, Mansoura University. His major research interests are particle swarm optimization (PSO), artificial intelligence (AI), genetic algorithms (GA), neural networks (NN), fuzzy logic, and deep learning (DL). Also, he is interested in the applications of AI in machine learning (ML), image processing, access control, and optimization.