Computers, Materials & Continua
DOI: 10.32604/cmc.2022.025353
Article

Tech Science Press

Embedding Extraction for Arabic Text Using the AraBERT Model

Amira Hamed Abo-Elghit[1,*], Taher Hamza[1] and Aya Al-Zoghby[2]

[1]Faculty of Computers and Information, Department of Computer Sciences, Mansoura University, Mansoura, 35516, Egypt
[2]Faculty of Computers and Artificial Intelligence, Department of Computer Sciences, Damietta University, Damietta, 34517, Egypt
*Corresponding Author: Amira Hamed Abo-Elghit. Email: amira-hamed@mans.edu.eg
Received: 21 November 2021; Accepted: 17 January 2022

Abstract: Nowadays, we can use the multi-task learning approach to train a machine-learning algorithm to learn multiple related tasks instead of training it to solve a single task. In this work, we propose an algorithm for estimating textual similarity scores and then use these scores in multiple tasks such as text ranking, essay grading, and question answering systems. We used several vectorization schemes to represent the Arabic texts in the SemEval2017-task3-subtask-D dataset. The used schemes include lexical-based similarity features, frequency-based features, and pre-trained model-based features. Also, we used contextual-based embedding models such as Arabic Bidirectional Encoder Representations from Transformers (AraBERT). We used the AraBERT model in two different variants. First, as a feature extractor in addition to the text vectorization schemes' features. We fed those features to various regression models to make a prediction value that represents the relevancy score between Arabic text units. Second, AraBERT is adopted as a pre-trained model, and its parameters are fine-tuned to estimate the relevancy scores between Arabic textual sentences. To evaluate the research results, we conducted several experiments to compare the use of the AraBERT model in its two variants. In terms of Mean Absolute Percentage Error (MAPE), the results show minor variance between AraBERT v0.2 as a feature extractor (21.7723) and the fine-tuned AraBERT v2 (21.8211). On the other hand, AraBERT v0.2-Large as a feature extractor outperforms the fine-tuned AraBERT v2 model on the used data set in terms of the coefficient of determination (R^2) values (0.014050, −0.032861), respectively.

Keywords: Semantic textual similarity; arabic language; embeddings; AraBERT; pre-trained models; regression; contextual-based models; concurrency concept

1 Introduction

The textual similarity is a critical topic in Natural Language Processing (NLP). That is due to its increasingly important turn in related topics such as text classification, recovery of specific information

This work is licensed under a Creative Commons Attribution 4.0 International License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

from data, clustering, topic retrieval, subject tracking, question answering systems, essay grading, and summarization. The textual similarity process tends to estimate the relevancy between text units [1,2]. The variations in the approaches existing in the literature review for textual similarity depend on the text representation scheme used before text comparison. Text representation is a significant task used to convert the unregulated form of textual data into a more formal construction before any additional text analysis or using it in predictive modeling [3]. Text representation, word embeddings, or vectorization means converting the text to numbers, which can be integers or floating-point values, then using it as input to machine learning algorithms [4]. We can divide the approaches of word embeddings into three categories: frequency-based or statistical-based, prediction-based or pre-trained, and contextual-based word embeddings. The frequency-based word embeddings approach is the traditional text modeling, which is based on the BOW representation. It contains One Hot Encoding (OHE), Hashing Vectorization, Part Of Speech (POS) Weighting [5], Word Counts, Term Frequency-Inverse Document Frequency (TFIDF) [4], and N-grams [6]. These vectorization techniques of text representation work well; however, they fail to keep a semantic relation between words or the meaning of a text, not considering the context in which a word appears. Consequently, the order of words' occurrence is lost as we create a vector of tokens in randomized order, and they may provide a sparse vector that contains a lot of zeros. The prediction-based or pre-trained word embedding models are trained on a large collection of texts to build fixed-length and continuous-valued vectors in low-dimensional space. The embedding size can vary depending on the target size selected during the training. It includes Word2Vec [7], Doc2Vec [8], FastText [9], GloVe [10], Aravec [11], etc. Pre-trained models save the time spent on obtaining, cleaning, and processing (intensively) enormous datasets. However, it, unfortunately, does not consider the relations between multiple words and the overall sentences' meanings or context within the text.

To overcome the above problems, contextual-based embedding models such as ELMo [12], ULMFiT [13], and BERT [14] are effective for learning complete sentence embeddings. They are used in sequence-level semantics learning of all the sequences in the documents. Thus, such models learn divergent embeddings for polysemous words. The ELMo model, e.g., is a dynamic language modeling technique to learn the embeddings of words based on context and considers the divergent embeddings of polysemous words. It contains two language models in each of the two directions that form a multilayer Recurrent Neural Network (RNN). The ULMFiT model is a left-to-right language model that boosts performance for some pre-trained models such as ELMo by including multiple fine-tuning techniques. By contrast to the ELMo, which incorporates the left-to-right and right-to-left models, the BERT uses bidirectional transformer training to provide more precise word embeddings. Three versions of BERT that address the Arabic language include the multilingual BERT (mBERT) [15] and two versions of AraBERT [16].

The main objective of this research is to propose an algorithm for estimating the textual similarity scores between Arabic texts. Then, use these scores in multiple tasks, such as text ranking, essay grading, and question answering systems. Our detailed objectives are: 1) Choosing the best text vectorization scheme to represent texts in the used dataset. 2) Picking the best regression model to make predictions represent the relevancy scores between text units from the applied regressors in terms of MAPE and R^2 Evaluation metrics. 3) Reducing the execution time of processing and increasing the CPU utilization as much as possible.

To implement our proposed algorithm, we used the AraBERT model in two different variants: first, we used it as a feature extractor model in addition to many other text embedding schemes such as word counts, TFIDF, and POS weighting as statistical-based approaches. Also, we use FastText and Aravec pre-trained models as prediction-based approaches. Then, we fed those features to several

regressors to make a prediction value that represents the relevancy score between their input texts. Second, we address the AraBERT model as a pre-trained model and fine-tune its parameters on the measuring textual similarity task to use the obtained results in many other tasks later.

The rest of this paper is organized as follows. The literature is reviewed in Section 2. Section 3 then describes the details of our proposed algorithm, and the experimental settings are introduced in Section 4. We present our experiments' details in Section 5. The discussion of results and implications is introduced in Section 6. Section 7 finally outlines the conclusion and suggests future work.

2 Review of Literature

Section 2.1 discusses the concept of textual similarity and the studies addressing it in the literature. Then, we address the recent research that used the AraBERT model in multiple NLP tasks in Section 2.2.

2.1 Textual Similarity and Its Approaches

In our previous work [1], we introduced a comprehensive overview of the textual similarity measurement approaches in the literature. We illustrated the differences between the categories of textual similarity concepts: lexical-based, semantic-based, and hybrid-based similarity in detail. We noticed that the differences in the approaches provided in the previous work depend on the text vectorization technique used before the text comparison process. There are various text vectorization techniques used, such as TFIDF, Latent Semantic Indexing (LSI) [17], and Graph-based Representation [18]. Due to these techniques, the similarity measure to compare text units differs because one similarity measure may not be convenient for all representation schemes. We summarized the most prominent attempts to measure the different textual similarity types and compared them according to the applied technique of feature extraction, the used dataset, and the results released by each approach. Then we shed light on the semantic analysis in the Arabic language, which is divided into four approaches: the word co-occurrence approach, the LSI approach, the feature-based approach, and the hybrid-based approach. Regarding the previous taxonomy mentioned above, we reviewed some of those approaches and summarized them according to the applied technique, the used dataset, the aim of each one, the similarity type (string-based, corpus-based, knowledge-based, or hybrid-based), and the results obtained by each approach.

Recently, [19] proposed a semantics-based approach for post-retrieval query-performance prediction depending on semantic similarities measured between entities in documents and queries. It consists of predictors for measuring semantic distinction, semantic query drift, and semantic cohesion in the top-ranked list of retrieved documents. The finding was that the proposed semantics approach is more effective in query performance predicting than the term-based methods by considering the semantic relatedness instead of the exact terms matching. They evaluated the proposed approach on the Robust04, ClueWeb09-B, and ClueWeb12-B datasets. Their queries' rankings are compared according to the proposed predictions and the actual values using Pearson and Kendall Correlation rank coefficients.

On the other hand, [20] proposed a probabilistic framework that incorporates Bidirectional Encoder Representations from Transformers (BERT) via sentence-level semantics into Pseudo-Relevance Feedback (PRF). They obtained the term importance at the term level. Then, they used the fine-tuned BERT model to get the embeddings of the query and the sentences in the feedback document to estimate the relevancy score between them. Next, the term scores at the sentence level are summed. Finally, the term-level and sentence-level weights are balanced by factors and combining

the top-k terms to generate a novel query for the next iteration of the processing. They conducted several experiments depending on six TREC datasets. As manifested by the evaluation indicators, the improved models outperformed the existing baseline models.

2.2 Using AraBERT Model in NLP Tasks

Several researchers have used the AraBERT model, either as a feature extractor model or by fine-tuning its parameters for a specific task. For example, [21] proposed three neural models: Bi-LSTM, CNN with FastText pre-trained word embeddings, and Transformer architecture with AraBERT embeddings. They are combined with three similarity measures for Arabic text similarity and plagiarism detection. They used the question similarity dataset for Semantic Textual Similarity (STS) called Mawdoo3 and the 2015 Arabic Pan dataset for plagiarism detection evaluation. Their results showed that the AraBERT-Transformer outperformed other models in terms of Pearson correlation with the Dot-Product-Similarity.

Reference [22] is another research that combined different types of classical and contextual embeddings: pre-trained word embeddings such as FastText and Aravec, pooled contextual embeddings, and AraBERT embeddings for processing Arabic Named Entity Recognition (NER) task on the AQMAR dataset. These embeddings are then fed into the Bi-LSTM. The experiments showed that the combination of the pooled contextual embeddings, FastText embeddings, and BERT embeddings had achieved the best performance. The proposed method in this research has achieved an F1 score of 77.62 percent, which outperforms all previously published results of deep and non-deep learning models on the same dataset.

Reference [23] paper addressed the pre-trained AraBERT model to learn complete contextual sentence embeddings to show its utilization in Arabic text multi-class categorization. They used it in two variants. The first is to transfer the AraBERT knowledge to the Arabic text categorization, and they fine-tuned the AraBERT's parameters on the OSAC datasets. Second, they used it as a feature extractor model, then fed its results to several classifiers, including CNN, LSTM, Bi-LSTM, MLP, and SVM. After comprehensive experiments, the findings showed that the fine-tuned AraBERT model accomplished state-of-the-art performance results (99%) in terms of F1-score and accuracy.

Reference [24] presented a binary classifier model to decide whether the pairs of verses provided by the QurSim dataset are semantically related or not. The AraBERT language model is used. They avoided redundancy and generated unrelated verse pairs from the QurSim dataset, dividing it into three datasets for comparisons. The experiments showed that the AraBERTv0.2 outperformed the AraBERTv2 on the three datasets in terms of accuracy score (92%).

Finally, [25] is shared in the EACL WANLP-2021 Shared Task 2: "Sarcasm and Sentiment Detection." and proposed a strategy consisting of two systems. The first system investigated whether a given Arabic tweet was sarcastic or not, which required performing deletions, segmentation, and insertion operations on different parts of the text. The other system aimed to detect the sentiment of the Arabic tweet from the ArSarcasm-v2 dataset that involved experimenting with multiple versions of two transformer-based models, AraELECTRA and AraBERT. They achieved the seventh and fourth places in the sarcasm and sentiment detection subtasks, respectively.

3 Methodology

This section extensively presents the methodology implemented for developing the proposed system. First, we start by describing the dataset used in this work. Then, we explain the proposed method and its modules.

3.1 Dataset

In this paper, we use the SemEval2017-task3 (Community Question Answering)-subtask-D (Rerank correct answers for a new question) dataset, which refers to the Arabic CQA-MD (Community Question Answering-Medical Domain) dataset [26]. It was collected from three Arabic medical websites (WebTeb, Altibbi, and Islamweb) that permit posting questions related to health and medical conditions by visitors and getting answers from professional doctors. It was divided into training, development, and testing datasets. Every dataset file includes a sequence of threads that begins with the original question and is associated with a list of 30 question-answer pairs, each with the following labels: D (Direct) means the QA pair contains a direct answer to the original question. R (Related) means the QA pair includes an answer to the original question that covers some of the aspects raised in the original question. In the end, I (irrelevant) means the QA pair contains an answer irrelevant to the original question.

Fig. 1 illustrates annotated questions from the dataset. Also, each QA pair is associated with some metadata, including the following: ID (QAID) is a unique ID of the question-answer pair. Relevance (QArel): the relevance of the question-answer pair concerning the question, which is to be predicted at test time, and Confidence (QAconf): this is the confidence value for the relevance annotation, based on inter-annotator agreement and other factors. This value is available for the training dataset only; it is not available for the development and test datasets.

```
The internal structure of a <Question> is the following:
<Question QID = "200670">
  <Qtext>ما هي العلاجات البديلة الموصى بها لعلاج مرض التصلب المتعدد </Qtext>
  <QApair QAID="20087" QArel="D" QAconf="0.373">
      <QAquestion>ماهي الكميه الموصى بها في عشبه سانت جون </QAquestion>
      <QAanswer> يجب استشارة الطبيب قبل تناول اي من الاعشاب لوصف الجرعة المناسبة حسب سبب الاستخدام</QAanswer>
    </QApair>
<QApair QAID="65318" QArel="I" QAconf="0.6642">
      <QAquestion> السلام عليكم ورحمة الله منذ ثلاث سنوات يتناول والدى (63 عام) دواء الكوجينتول لعلاج انفصام الشخصية فأصبحت حالته
جيده خلال تلك الفترة واختفت الاعراض لكن فى الفترة الاخيرة ظهرت عليه الاعراض الجانبية للدواء وهى اعراض شديدة فهل الحل الوحيد تخفيف
الجرعة او ايقاف التناول وهل هناك ادوية بديله افضل وماهى الجرعة الموصى بها ولكم جزيل الشكر </QAquestion>
      <QAanswer>تم إجابة السؤال من قبل</QAanswer>
    </QApair>
<QApair QAID="26550" QArel="R" QAconf="0.7038">
      <QAquestion> ماسبب بروز اوردة زرقاء في جبهة وجهي وفي قشرة راسي كلما ضحكت برزت بوضوح وكانها تريد ان تنفجر مع العلم اني مصابة
بمتلازمة كريست هل ممكن علاجها او التخفيف من مضهرها بالادوية </QAquestion>
      <QAanswer>هو مرض نادر يصيب النسيج الضام; وسبب تسارع في تنسج (SCHLERODERMIA) متلازمة كرست او مرض التصلب الجلدي
الكيراتين مما يضفي على الجلد صلابة مفرطة; كذلك يسبب اعتلال (تصلب و تضيق) الاوعية الدموية الصغيرة تحت الجلد; وفي بعض الاحيان يتقاطع
مع امراض في الاجهزه الاخرى. </QAanswer>
    </QApair>
</Question>
Where, <Qtext> is the text of the question.
<QApair> is a question answer pair retrieved using a search engine. There are about 30 instances of <QApair> per
<Question>.
```

Figure 1: Annotated question from the Arabic CQA-MD dataset

So, we use this dataset (training and development) with this associated metadata to accomplish our primary research objective: estimate the relevancy scores between text pairs. We consider the confidence (QAconf) values as the relevancy score between the question and its QA pairs.

3.2 Text Preprocessing Phase

In Fig. 2, we propose two models for preprocessing: simple preprocessing and full preprocessing, depending on the nature of the task of the subsequent phases. For instance, we only need some preprocessing steps to transform data into a form that matches the AraBERT model, such as removing diacritics, punctuations, and URL text. So, we consider this situation in our proposed methodology and define two types of preprocessing steps. The simple preprocessing procedure includes diacritics removal (Tashkeel_Removing Function), punctuations, URL text removal, and spell checking. Then, we apply the tokenization task to split the text into its tokens using the AraBERT tokenizer. Afterward, we change each text to a BERT format by adding the particular [CLS] token at the start of each text and a [SEP] token between the sentence and the end. Then, we determine each token's index according to AraBERT's vocabulary. The full preprocessing contains the same steps as the previous preprocessing type, in addition to stopwords removing, named entity recognition (NER), stemming, and lemmatization tasks, respectively. But there is a difference between the tokenization task in both algorithms.

To complete the diacritics removing task (Tashkeel_Removing function), we use the Tashaphyne Python library [27], which is an Arabic light stemmer and segmentor. In the normalize module of this package, we specifically use the function strip_tashkeel. We define a set of patterns to detect any punctuation symbols and URL text in the text using the re python library, which provides several functions to facilitate the search for a specific pattern or string form in the text and remove it from the text. Next, we use Farasa [28], an Arabic Natural Language Processing (ANLP) toolkit serving the spellchecking task and several other tasks such as segmentation, stemming, Named Entity Recognition (NER), and part-of-speech tagging. As shown in Algorithm 1 of the full preprocessing, we use a built-in function in Python called split that allows changing the default splitter from space to any symbol or character if we need it. Consequently, we remove them from sentences using the Natural Language Toolkit (NLTK) Python package, which includes a stopwords corpus containing stopwords' lists for Arabic and many other languages [29]. The Farasa Named Entity Recognizer is used to generate a list of named entities in text. The aim behind using this technique as a step of preprocessing steps is to keep the named entities found in a text without any change that may be happening to them in the stemming task, as shown in Tab. 1.

Arabic stemmers are categorized into two categories: light-based stemmers and root-based stemmers. This type is used in the stemming step with the Farasa Stemmer web API. Also, we use Khoja stemmer as a root-based stemmer [30]. Consequently, after applying the NER and stemming processes to a text, we compare the output list from the NER process to the output list from the stemming process to obtain the final representation of the given text, as shown in Fig. 3. Thus, we are given a set of questions, each of which is associated with a set called P that includes question-answer pairs. To compute our features, we define a question with its question-answer pairs as $< T1, T2 >$, where T1 is the original question and T2 is a question from its question-answer pair according to three setups:

- Simple processed data setup in which we perform simple preprocessing on T1 and T2 before using them in the AraBERT model.
- Stemmed data setup in which the stemming process from the full preprocessing phase is applied to T1 and T2.
- Lemmatized data setup in which the lemmatization process from the full preprocessing phase is applied to T1 and T2.

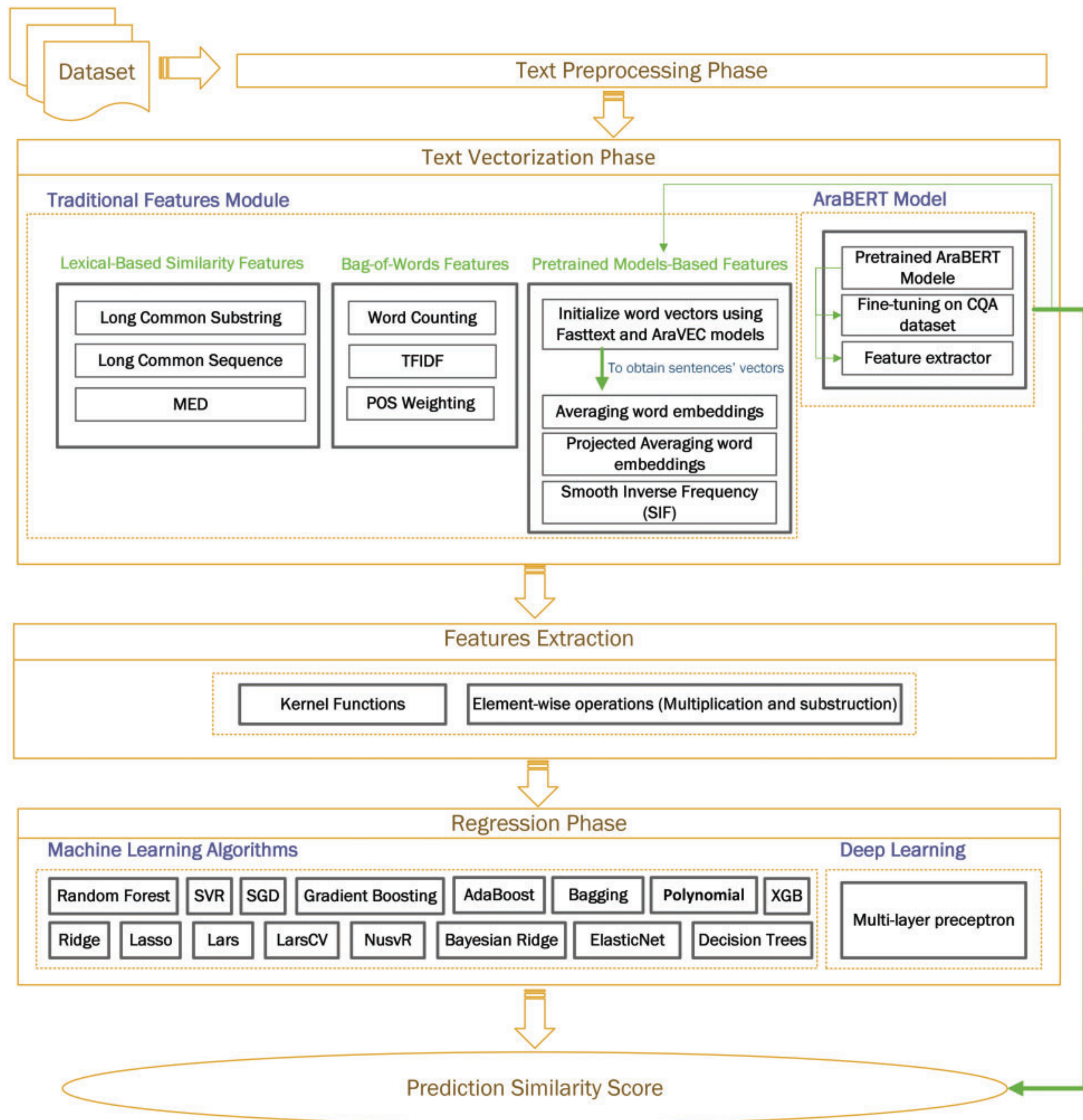


Figure 2: System architecture

Algorithm 1: Full Text Preprocessing

1. Function: Full_Preprocessing (q_i,p_i)
2. Input:
3 q_i: question
4 p_i: list of approximately 30 pair of answers retrieved for q_i
5. Output:
6. stemmed_ques: string object represents stemmed version of preprocessed question // initially null
7. lemmatized_ques: string object represents lemmatized version of preprocessed question// initially null
8. stemmed_pairs: preprocessed list of stemmed answers for this question // initially empty
9. lemmatized_pairs: preprocessed list of lemmatized answers for this question // initially empty
10. Variables:
11. q_o: preprocessed question
12. p_o: a list of preprocessed answers for q_o
13. token_list: list of tokens, initially empty
14. tokens_after_stopwords_remove: list of tokens after removes stopwords from them // initially empty
15. ner_list: list of tuples of NER process // initially empty
16. stems_list: list of stems of tokens // initially empty
17. stemmed_ques: string object represents the stemmed version of the question
18. lemmatized_ques: string object represents the lemmatized version of the question
19. stemmed_pairs: preprocessed list of stemmed answers for this question
20. lemmatized_pairs: preprocessed list of lemmatized answers for this question
21. Begin
22. q_o = Tashkeel_Removing (q_i)
23. q_o = URL_Removing (q_i)
24. q_o = Punctuation_Removing (q_i)
25. q_o = Spellchecking (q_i)
26. tokens_list = Tokenization (q_i)
27. tokens_after_stopwords_remove = Stopwords_Removing (tokens_list)
28. ner_list = Named Entity Recognition (tokens_after_stopwords_remove)
29. stems_list = Stemming (tokens_after_stopwords_remove)
30. stemmed_ques = Compare (ner_list, stems_list)
31. lemmatized_ques = Lemmatize (stemmed_ques)
32. for answer in p_i answer = 0, 1, ... do
33. p_o [answer] = Tashkeel_Removing (p_i [answer])
34. p_o [answer] = URL_Removing (p_i [answer])
35. p_o [answer] = Punctuation_Removing (p_i [answer])

(Continued)

Algorithm 1: Continued

```
36.       p_o [answer] = Spellchecking (p_i [answer])
37.       token_list = Tokenization (p_i [answer])
38.       tokens_after_stopwords_remove = Stopwords_Removing (token_list)
39.       ner_list = Named Entity Recognition (tokens_after_stopwords_remove)
40.       stems_list = Stemming (tokens_after_stopwords_remove)
41.       stemmed_pairs. insert (answer, Compare (ner_list, stems_list ))
42.       lemmatized_pairs. insert (answer, Lemmatize (stems_list ))
43.       return stemmed_ques, lemmatized_ques, stemmed_pairs, lemmatized_pairs
44. End
```

Table 1: Representation of the influence of the spell checking and NER processes on stemming and lemmatization results

The raw sentence:	مرض باركيتسون من الامراض اللتى تحدث فى كبار السن غالبا بعد ال 60عاما لنقص فى مادة الدوبامين فى المخ مما ينتج عنها رعشة بالاطراف
After Punctuations Removing without Spellchecking:	مرض باركيتسون من الامراض اللتي تحدث في كبار السن غالبا بعد ال عاما لنقص في مادة الدوبامين في المخ مما ينتج عنها رعشة بالاطراف
After Stopwords Removing without Spellchecking:	مرض باركيتسون الامراض اللتي تحدث كبار السن غالبا ال عاما لنقص مادة الدوبامين المخ ينتج عنها رعشة بالاطراف
After Stemming process without Spellchecking and NER processes:	مرض باركيتسون امراض لتي تحدث كبير سن غالب عام نقص مادة دوبامين مخ أنتج عن رعش اطراف
After Lemmatization process without Spellchecking and NER processes:	مرض باركيتسون مرض ل حدث كبر وسن غلب عوم نقص ميد دوبامين خ نتج رعش طرف
Sentence with Spellchecking:	مرض باركينسون من الأمراض التي تحدث في كبار السن غالبا بعد ال عاما لنقص في مادة الدوبامين في المخ مما ينتج عنها رعشة بالأطراف
After Stopwords Removing:	مرض باركينسون الامراض التي تحدث كبار السن غالبا ال عاما لنقص مادة الدوبامين المخ ينتج عنها رعشة بالأطراف
After NER process:	[('مرض', 'O'), ('باركينسون', 'B-LOC'), ('الامراض', 'O'), ('التي', 'O'), ('تحدث', 'O'), ('كبار', 'O'), ('السن', 'O'), ('غالبا', 'O'), ('ال', 'O'), ('عاما', 'O'), ('لنقص', 'O'), ('مادة', 'O'), ('الدوبامين', 'O'), ('المخ', 'O'), ('ينتج', 'O'), ('عنها', 'O'), ('رعشة', 'O'), ('بالأطراف', 'O')]
After Stemming process with Spellchecking and NER processes:	مرض باركينسون مرض التي تحدث كبير سن غالب عام نقص مادة دوبامين مخ أنتج عن رعش طرف
After Lemmatization process with Spellchecking and NER processes:	['مرض', 'باركينسون', 'مرض', 'حدث', 'كبر', 'وسن', 'غلب', 'عوم', 'نقص', 'ميد', 'دوبامين', 'خ', 'نتج', 'رعش', 'طرف']
After compare NER and Lemmatization processes' results	مرض باركينسون مرض حدث كبر وسن غلب عوم نقص ميد دوبامين خ نتج رعش طرف

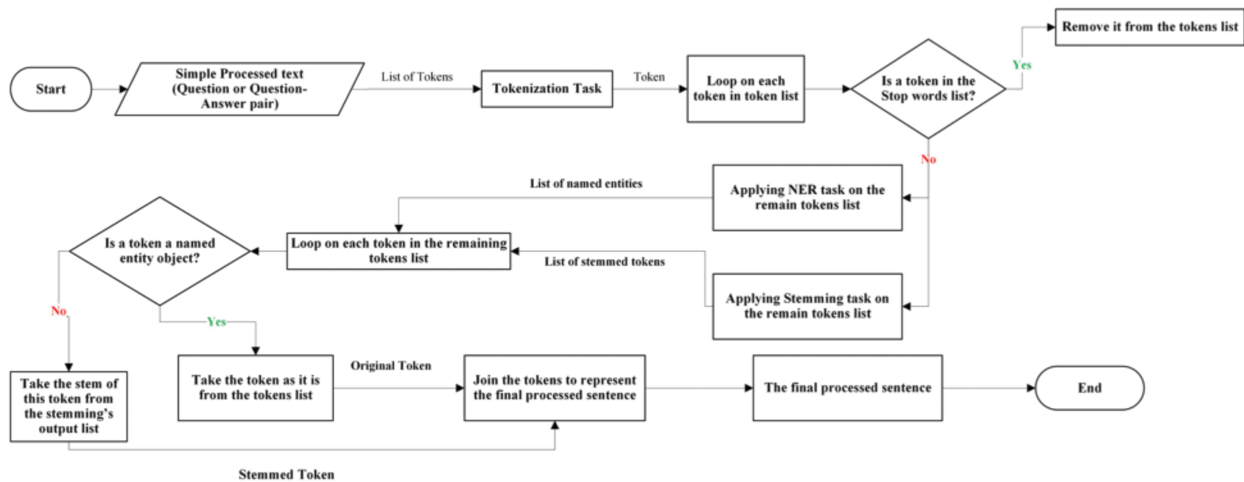


Figure 3: Comparison between NER and stemming processes' results

3.3 Text Vectorization Phase

This phase consists of two modules: the traditional features module and the AraBERT model.

3.3.1 Traditional Features Module

In this phase, we execute multiple feature engineering techniques. To begin, we employ three-sentence pair matching metrics: Long Common Substring/Sequence [31], Levenshtein distance, and Minimum Edit Distance (MED) [32], which are intended to directly calculate the similarity (overlapping of characters/terms/substrings) of two sequences. To obtain an accurate sequence similarity value, the stopwords are removed and each word is lemmatized. Consequently, for each sentence pair, we get three features as lexical-based similarity features. Second, we apply three types of statistical-based embedding techniques: word counting, TFIDF, and POS weighting. We use the sklearn.feature_extraction module [33] to extract these features in a format reinforced by machine learning algorithms from the dataset. We consider the sparsity problem, which may be caused by them, and try to solve it by applying these steps in the preprocessing phase: stopwords from being removed, fixing misspelt words, and reducing words to their lemma.

To ensure that these steps have an effect, we calculate the number of vocabulary in our dataset; it is approximately 21835 in the stemmed data setups and it becomes approximately 10988 in the lemmatized data setup. We notice that the number of vocabulary words decreased using the lemmatization process. Consequently, the dimensionality of vectors decreases. Third, we apply some pre-trained word embedding models: FastText and Aravec, both of which apply to the Arabic language. Tab. 2 shows the versions of pre-trained word embedding that were used in this study. In Python, we use the Genism library that provides access to FastText and other word embedding algorithms for training and extracting word vectors; it allows us to download pre-trained models from the internet to be loaded and fine-tuned [34].

We try each of these models individually to initialize word embedding, although we sometimes cannot find embeddings for some words in a sentence. Consequently, we combined them to complete each other and obtain a large number of word embeddings. Because of the nature of our used dataset, a sentence may contain foreign words that are the names of medicines or diseases that are not found in

the Aravec or Arabic Fast text models. To address this issue, we used the FastText model's multilingual model, as shown in Fig. 4. We notice that some of the words are misspelled, so we cannot use an embedding model to get the correct embedding for them. Thus, spell checking is an essential step in the preprocessing phase. Second, we must convert some words to their lemma form to get their embeddings from pre-trained models such as the Aravec model. However, there are some terms that are not found in an embedding model even after correcting them; thus, we ignore them from a sentence.

Table 2: Pre-trained word embedding models

Embeddings model' name	Dimension	Source
FastText	300d	fastText-wiki-news-subwords 300
Arabic_FastText	300d	cc.ar.300.vec
Aravec	300d	full_grams_sg_300_wiki.mdl

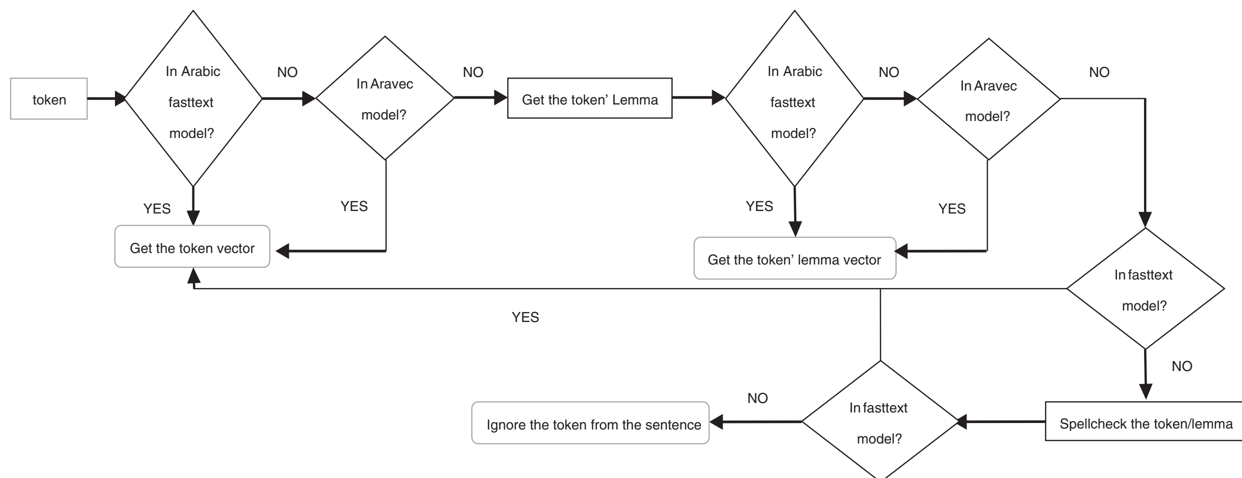


Figure 4: Process of obtaining word embeddings from the pre-trained models

To obtain a single sentence vector representing the embedding of each sentence, we adopt several methods, such as averaging the word vectors that form each sentence, the averaged vectors that result are multiplied by a projection matrix, and using smooth inverse frequency (SIF) [35] to estimate each word-embedding weight by a / (a + p(w)), where a is a parameter that is typically set to 0.001 and p(w) is the frequency of the word in a dataset, contrary to the previous methods that assign equivalent weights to each word in the sentence.

3.3.2 AraBERT Model

Because we are dealing with Arabic texts, we use the AraBERT model, which is an Arabic pre-trained language model based on the BERT architecture [14,16]. There are four releases of it: AraBERT v0.1, AraBERT v1, AraBERT v0.2, and AraBERT v2. They may differ from each other in using the Farasa segmenter that will split affixes from the text. AraBERT now comes in four new variants. All models are accessible on the HuggingFace model page under the aubmindlab name. AraBERT models are pre-trained on a massive collection of text and then fine-tuned for different tasks. Consequently, we used the AraBERT model in two ways: first, we investigated and fine-tuned its

parameters for the Semantic Textual Similarity (STS) task, and then, we fed the AraBERT embeddings to a feed-forward layer containing one neuron with a linear activation function to predict the similarity scores. Second, we apply it as a feature extractor model to obtain a fixed-length tensor (usually 768 for AraBERT Base models and 1024 for AraBERT Large models). To obtain sentence embeddings, the average pooling of all tokens' layers is estimated. Then, feed these obtained embeddings to the regression models. In two variants, we compare the AraBERT model with the Multilingual BERT (mBERT) model [15].

3.4 Features Extraction Phase

In this phase, we use two methods to extract features from each sentence pair's vectors: kernels and element-wise operations. To begin with, we want to maintain the discriminating power of lexical-based similarity features when compared to the dimensionality of a vector derived from each BOW feature for each sentence. Consequently, we estimate sentence pair distances using 12 kernel functions and combine them with lexical-based similarity features to represent each sentence pair. Tab. 3 shows the 12 kernel functions that were used in this work. We notice that these features are on different scales, which may have an impact on the fit of regression models in the following phase. Thus, we attempt to normalize them into [0, 1] using the max-min normalization technique and standardize them around 0 using the StandardScaler module before building regression models.

Table 3: Used kernel functions

Type	Measurements
Linear kernels	Cosine distance, Manhattan distance, Euclidean distance, Chebyshev distance, Linear distance
Stat kernels	Pearson coefficient, Spearman coefficient, Kendall tau coefficient
Non-linear kernels	Polynomial, RBF, Laplacian, sigmoid

The second method is the element-wise operations that contain a large category of operations such as arithmetic, comparison, and other operations that operate on corresponding elements within the respective tensors or vectors. For each sentence pair, we use two types of operations: multiplication and subtraction. Then, we concatenate the results into a single tensor.

3.5 Regression Phase

Different machine learning algorithms and deep learning models are considered for building regression models to make predictions that represent textual similarity scores.

3.5.1 Machine Learning Regression Algorithms

We investigate multiple learning algorithms for regression tasks such as Random Forest (RF), Support Vector Regressor (SVR), Nu Support Vector Regression (NuSVR), Gradient Boosting (GB), AdaBoost, least angle regression (LARS), Cross-validated Least Angle Regression (LARSCV), Bagging regressor, Stochastic Gradient Descent (SGD), Ridge regressor, Bayesian Ridge regressor, Decision Trees, Lasso regressor, Elastic Net, Polynomial regressor, and Extreme Gradient Boosting (XGB) [36]. In Python, we use the scikit-learn toolkit [37] to implement these algorithms except for the XGB regressor, which we used the xgboost package to implement in our work.

3.5.2 Deep Learning Models

We implement a multilayer perceptron model (MLP) that comprises two hidden layers with the ReLU activation function. Sentence pairs' embeddings represent the input that is fed to these layers. The first hidden layer contains the number of input dimensions plus 50 hidden neurons, i.e., if the dimensions of the input embeddings are equal to (600,), the number of input dimensions plus 10 hidden nodes comprises the second hidden layer. This should be noted. We experimented with wider and deeper neural networks in this model, but the wider neural network outperformed the deeper neural network in the experiments; hence, we rely on this neural network architecture. We use Adam [38] as an optimization technique and use Mean Square Error (MSE) and Mean Absolute Error (MAE) as both loss and evaluation functions [39]. Then, we set the validation split parameter to 0.2; hence, 80% of the data is used to train the model, whereas the remaining 20% is used for testing purposes, with epochs of 100 and batch_size of 100. Finally, the output layer contains one hidden neuron with a linear activation function to make a prediction. In Python, we use the Keras API, based on the TensorFlow and Theano [40] packages, for executing high-level neural networks; thus, we needed to have TensorFlow installed on our system first.

4 Experimental Settings

We describe the concurrency concept in Section 4.1, and the evaluation metrics are described in Section 4.2.

4.1 Concurrency Concepts

Section 3.1 shows that the answer pairs' total number is enormous and needs a long time to process all these pairs, which may be up to several days. For the experiment, we selected a sample that includes eight training questions with their pairs to be preprocessed. The time taken in this experiment is 1 hr: 23 m: 48 s. It has been a long time, so to speed up program execution on this dataset, we used the concurrency concept, which is about parallel computation. All our experiments were run on a CPU processor with four cores using the Python Interpreter 3.8. There are three types of concurrency concepts: multithreading [41] is also known as preemptive multitasking, as the OS knows about each thread and can interrupt at any moment to start executing on another thread. Second, Asyncio [42] is also referred to as cooperative multitasking because the tasks collaborate and decide when to relinquish control. Finally, multiprocessing [43] achieves true concurrent execution because the processes run concurrently on different processors on different CPU cores. We will only look at two types: multithreading and multiprocessing.

ThreadPool is a technology for achieving concurrency of execution in a computer program. It keeps a pool of idle threads pre-instantiated and ready to be assigned tasks; hence, it eliminates the creation time required to create them one by one. Another advantage of thread pools is that a thread can be reused once its execution is complete. We use concurrent.futures, a Python standard library module that includes a concrete subclass known as ThreadPoolExecuter, which uses multithreading, and we get a pool of threads for submitting the tasks. The pool thus created assigns tasks to the available threads and arranges them to run. For applying the multiprocessing concept in Python, we use a multiprocessing library for creating multiprocess operations, in which the process class creates a queue object to store the results of each process, in which the Queue class. The multiprocessing module provides the pool class, which offers a convenient means of parallelizing the execution of a function across multiple input values, distributing the input data across processes. There is no guarantee that multithreaded will be faster because it depends on the type of program; there is a

performance difference between CPU-bound and I/O-bound programs. When the tasks are CPU-intensive, we should consider the multiprocessing module. By contrast, the tasks are I/O bound and require plenty of connections; the multithreading concept is recommended. To demonstrate that the multithreading concept is best suited for our I/O bound program, we ran several experiments on different dataset samples.

4.1.1 Experiments 1 and 2

Table 4: Comparison between sequential computation/running and the two concepts of parallel computation in experiments 1 and 2

	Experiment 1			Experiment 2		
	Sequential computation	Parallel computation		Sequential computation	Parallel computation	
		Multi-threading	Multi-processing		Multi-threading	Multi-processing
#Num of processes	1 process	1 process	5 processes, each work on 2 questions	1 process	1 process	4 processes, each work on 2 questions
#Num of threads	-	5 threads, each work on 2 questions	-	-	4 threads, each work on 2 questions	-
#Num of questions	10 Questions with their pairs	10 Questions with their pairs	10 Questions with their pairs	8 Questions with their pairs	8 Questions with their pairs	8 Questions with their pairs
CPU utilization	average, but sometime be very low	Very high and most of times achieve 100%	Very high and most of times achieve 100%	average, but sometime be very low	Very high and most of times achieve 100%	Very high and most of times achieve 100%
Taken time	1 h: 45 m: 48 s	55 m: 10 s	55 m: 31 s	1 h: 23 m: 48 s	44 m: 17 s	43 m: 10 s

As shown in Tab. 4, both experiments 1 and 2 are applied to the first 8–10 questions with their pairs in the training dataset. In Fig. 5, we observe the following: In both experiments, sequential computation took more execution time than parallel computation. Hence, we eliminate sequential computation in the following experiments. The two types of parallel computation outperform sequential computation, but the difference in execution time between multithreading and multiprocessing is trivial.

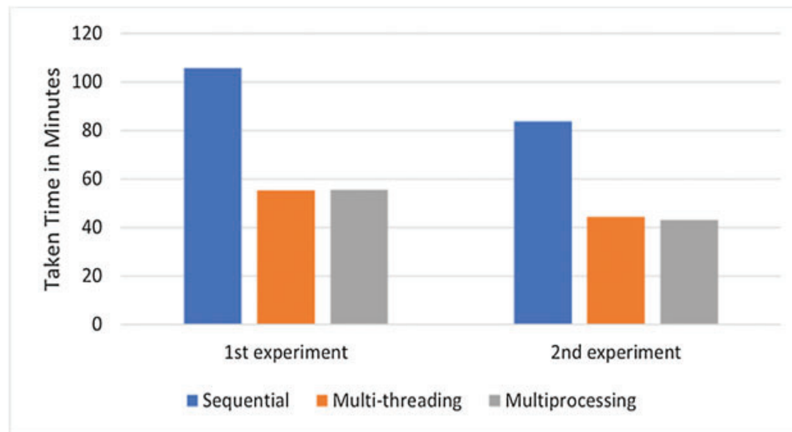


Figure 5: Comparison between running time of sequential computation and the parallel computation

4.1.2 Experiments 3 and 4

Table 5: Comparison between sequential computation/running and the two concepts of parallel computation in experiments 3 and 4

	Experiment 3			Experiment 4		
	Parallel computation					
	Multi-processing	Multi-threading	Multi-processing with multi-threading (Hybrid)	Multi-processing	Multi-threading	Multi-processing with multi-threading (Hybrid)
#Num of processes	5 processes, each work on 2 questions	1 process	5 processes, each with 2 threads	7 processes	1 process	5 processes, each with 2 threads
#Num of threads	-	5 threads, each work on 2 questions	10 threads, each work on 1 question (but 6 threads worked).	-	7 threads	10 threads, each work on 1 question (but 7 threads worked).
#Num of questions	10 Questions with their pairs	10 Questions with their pairs	10 Questions with their pairs (but 6 questions only processed).	7 Questions with their pairs, each process work on 1.	7 Questions with their pairs, each thread work on 1.	10 Questions with their pairs (but 7 questions only processed).
CPU utilization	Very high and most of times achieve 100%	Very high and most of times achieve 100%	Very high and most of times achieve 100%	Very high and most of times achieve 100%	Very high and most of times achieve 100%	Very high and most of times achieve 100%
Taken time	00 h: 55 m: 31 s	51 m: 57 s	00 h: 35 m: 26 s	00 h: 49 m: 38 s	00 h: 44 m: 20 s	00 h: 52 m: 28 s

As shown in Tab. 5, in both experiments 3 and 4, we apply multithreading and multiprocessing concepts and try to emerge with a hybrid concept, trying to decrease executing time. From the training dataset, we take a sample of 10 questions and their pairs, ranging from index 76 to index 86. However, only 6–7 questions were processed in the hybrid concept. In the fourth experiment, we hope to explain why only six of the 10 questions in the third experiment worked. The hybrid concept comes first in this experiment's order. Then both other concepts are applied to the same number and order of questions that are processed first in the hybrid. We determined that 6–7 threads only worked because the total number of threads that can run on our CPU processor with its cores is eight threads if all cores are occupied. Thus, the peak number of threads that can be run is equal to or less than eight, on the condition that no other programs or processes are running. In Fig. 6, we observe the following: The peak number of threads that can run on all CPU cores is not constant. It changes from one experiment to another. Hybrid concepts take a longer execution time when compared with multithreading and multiprocessing individually. We prove that the multithreading concept is the most suitable for our task as it takes the least amount of time to execute.

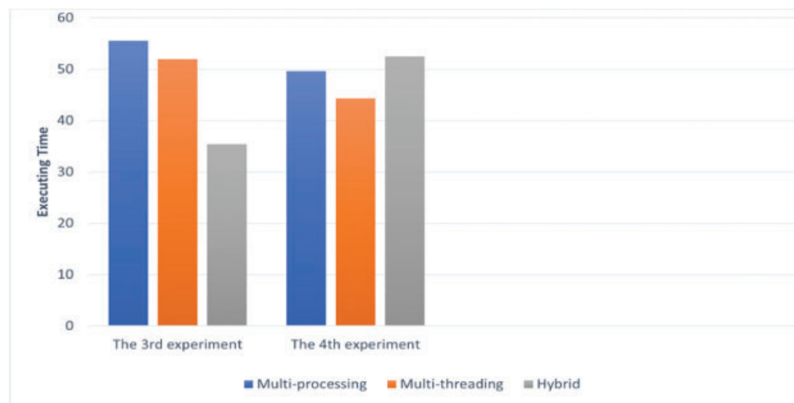


Figure 6: Comparison between running time of sequential computation and the parallel computation

4.2 Regression Models Evaluation Metrics

We used different metrics to evaluate the performance of different regression models on different types of features, as listed below:

Root Mean Square Error (RMSE): It is the squared root of the mean of summation of squared prediction error as shown in Eq. (2) [39]. The prediction error of a row of data is shown in Eq. (1). It converts the value of errors back to the units of the output variable, which makes it meaningful for interpretation. Its value varies from 0 to ∞. A value of 0 indicates a perfect fit; the smaller the value, the better the fit.

$$\text{Prediction Error} = \text{Actual Value} - \text{Predicted Value} \tag{1}$$

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \widehat{y}_i)^2} \tag{2}$$

Mean Absolute Error (MAE): It is the mean of the summation of absolute squared prediction errors as shown in Eq. (3). Compared to RMSE, MAE is robust to the presence of outliers because it

uses the absolute value. A value of 0 indicates a perfect fit; the smaller the value, the better the fit [39].

$$\frac{1}{N}\sum_{i=1}^{N}|y_i - \widehat{y_i}| \tag{3}$$

The coefficient of determination (R^2 score) is the square of the correlation coefficient (R). It determines how well the regression predictions explore the real data points as shown in the equation as shown in Eq. (4).

$$R^2 = 1 - \frac{\sum_{i=1}^{N}(y_i - \widehat{y_i})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2} \tag{4}$$

where $\widehat{y_i}$ represents the prediction value of y_i and $\bar{y}$ is the mean of actual data.

$$\bar{y} = \sum_{i=1}^{N} y_i \tag{5}$$

R^2 varies from 0 to 1. A value of 1 indicates that the regression predictions perfectly fit the data [39]. 0 indicates that the model does not explain the variability of the response data around its mean. R^2 may be a negative value when the model selected does not appropriately represent the nature of the data.

The Mean Absolute Percentage Error (MAPE) is a popular metric for assessing generic regression problems [44]. It is given by the following formula, as shown in Eq. (6). We can multiply this formula by 100% to express the number as a percentage.

$$M = \frac{1}{n}\sum_{t=1}^{n}\left|\frac{y_i - \widehat{y_i}}{y_i}\right| \tag{6}$$

5 Experiments and Results

The experimental results are analyzed and shown in the following. A comparison between using stemmed and lemmatized data setups is discussed in Section 5.1. The frequency-based with lexical-based features experiment is described in Section 5.2. The pre-trained models' experiment is described in Section 5.3. AraBERT as a feature-extracting model experiment is discussed in Section 5.4. Then, we compared the findings of the MLP model in terms of MAE and MAPE with the best regressors of all the previous experiments.

5.1 Stemmed and Lemmatized Data Setups Experiment

In this experiment, we aim to determine if using lemma to decrease the dimensionality of the vector represented by BOW (word counting, TFIDF, and POS weighting) features provides better outcomes in the regression phase or not. Then, in the regression phase, we identify the influence of lemmatization and stemming processes on the data as we try these two data setups to test the quality of them as indicated by the results of the regressor model, which are evaluated via the RMSE as shown in Eq. (2). We discard the SGD regressor because its results show that it does not properly explore the problem variables. We select the minimum average of the RMSE values of regression models with different features (word counting, TFIDF, and POS weighting) as shown in Fig. 7. We conclude the following:

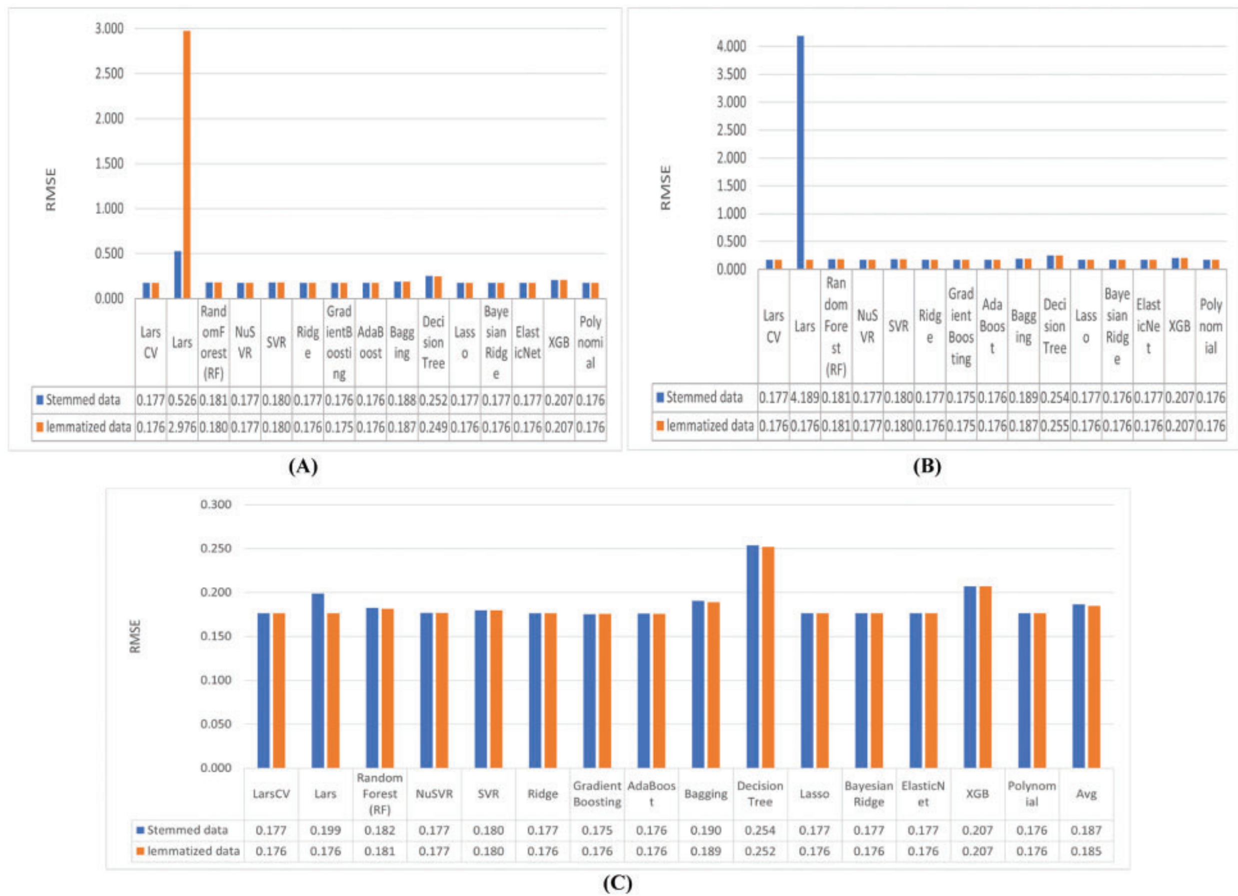


Figure 7: Using stemmed and lemmatized data setups: (A) that represents using these data setups on word counting representation, (B) that represents using these data setups on TFIDF representation, and (C) that represents using these data setups on POS weighting representation

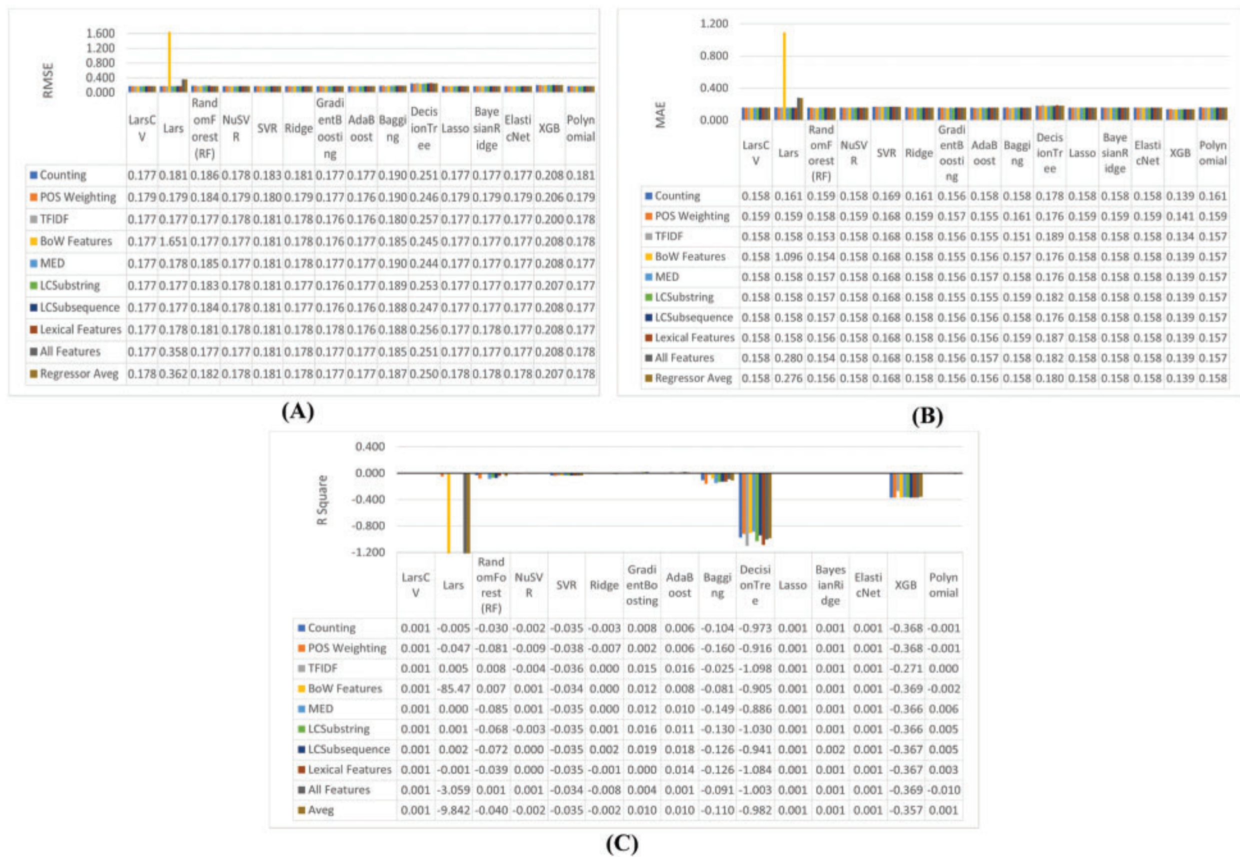
In the word counting representation, the average of the RMSE values of regression models according to stemmed data setup equals 0.208107, and lemmatized data setup equals 0.371046067. Hence, we observe that the stemmed data setup is more appropriate for word counting than the lemmatized data setup. In the TFIDF representation, the average of the RMSE values of regression models according to stemmed data setup equals 0.452521867, and the lemmatized data setup equals 0.1848484. Hence, we observe that the lemmatized data setup is more appropriate for the TFIDF than the stemmed data setup. In the POS weighting representation, the average of the RMSE values of regression models according to stemmed data setup equals 0.186642, and the lemmatized data setup equals 0.184832333. We realize that the difference between the two values is not particularly significant. Hence, the POS weighting features are neutral, indicating that their influence is minor in comparison to other features, as demonstrated by the following experiments.

5.2 Frequency/Lexical Based Features' Experiment

After applying kernel functions to the vectors obtained from each BOW feature for each sentence pair, with the purpose of keeping the discriminating power of lexical-based similarity features compared with them. In this experiment, we intend to filter which are the best regressors based on

both BOW and lexical-based features to select the best pool of regressors and the best types of features, where the BOW feature represents the word counting, POS weighting, and TFIDF features collectively. MED, Long Common Substring, and Long Common Subsequence are all represented by lexical-based features. All features represent all BOW and lexical-based features together, as shown in Fig. 8. The filtering process is based on selecting the smallest value in both RMSE and MAE and the highest value in the R^2 metric. Then, in ascending order, we rank those different selected models based on MAE metric values, which is the best metric for this purpose because it does not reflect large residuals. According to Fig. 9, we conclude the following:

- We notice the XGP regressor gives the best values in terms of MAE and MAPE. However, the values of R^2 are negative, so we do not select it.
- The best regressors are the Gradient Boosting, AdaBoost, and Lasso regressors.
- The best representation schemes for features are Long Common Substring, Long Common Subsequence, and BOW features (word counting, TFIDF, and POS weighting together).



	LarsCV	Lars	RandomForest (RF)	NuSVR	SVR	Ridge	GradientBoosting	AdaBoost	Bagging	DecisionTree	Lasso	BayesianRidge	ElasticNet	XGB	Polynomial
Counting	0.177	0.181	0.186	0.178	0.183	0.181	0.177	0.177	0.190	0.251	0.177	0.177	0.177	0.208	0.181
POS Weighting	0.179	0.179	0.184	0.179	0.180	0.179	0.177	0.176	0.190	0.246	0.179	0.179	0.179	0.206	0.179
TFIDF	0.177	0.177	0.177	0.178	0.181	0.178	0.176	0.176	0.180	0.257	0.177	0.177	0.177	0.200	0.178
BoW Features	0.177	1.651	0.177	0.177	0.181	0.178	0.176	0.177	0.185	0.245	0.177	0.177	0.177	0.208	0.178
MED	0.177	0.178	0.185	0.177	0.181	0.178	0.177	0.177	0.190	0.244	0.177	0.177	0.177	0.208	0.177
LCSubstring	0.177	0.177	0.183	0.178	0.181	0.177	0.176	0.177	0.189	0.253	0.177	0.177	0.177	0.207	0.177
LCSubsequence	0.177	0.177	0.184	0.178	0.181	0.177	0.176	0.176	0.188	0.247	0.177	0.177	0.177	0.208	0.177
Lexical Features	0.177	0.178	0.181	0.178	0.181	0.178	0.178	0.176	0.188	0.256	0.177	0.178	0.177	0.208	0.177
All Features	0.177	0.358	0.177	0.177	0.181	0.178	0.177	0.177	0.185	0.251	0.177	0.177	0.177	0.208	0.178
Regressor Aveg	0.178	0.362	0.182	0.178	0.181	0.178	0.177	0.177	0.187	0.250	0.178	0.178	0.178	0.207	0.178

(A)

	LarsCV	Lars	RandomForest (RF)	NuSVR	SVR	Ridge	GradientBoosting	AdaBoost	Bagging	DecisionTree	Lasso	BayesianRidge	ElasticNet	XGB	Polynomial
Counting	0.158	0.161	0.159	0.158	0.169	0.161	0.156	0.158	0.158	0.178	0.158	0.158	0.158	0.139	0.161
POS Weighting	0.159	0.159	0.158	0.159	0.168	0.159	0.157	0.155	0.161	0.176	0.159	0.159	0.159	0.141	0.159
TFIDF	0.158	0.158	0.153	0.158	0.168	0.158	0.156	0.155	0.151	0.189	0.158	0.158	0.158	0.134	0.157
BoW Features	0.158	1.096	0.154	0.158	0.168	0.158	0.155	0.156	0.157	0.176	0.158	0.158	0.158	0.139	0.157
MED	0.158	0.158	0.157	0.158	0.168	0.158	0.156	0.157	0.158	0.176	0.158	0.158	0.158	0.139	0.157
LCSubstring	0.158	0.158	0.157	0.158	0.168	0.158	0.155	0.155	0.159	0.182	0.158	0.158	0.158	0.139	0.157
LCSubsequence	0.158	0.158	0.157	0.158	0.168	0.158	0.156	0.156	0.158	0.176	0.158	0.158	0.158	0.139	0.157
Lexical Features	0.158	0.158	0.156	0.158	0.168	0.158	0.156	0.156	0.159	0.187	0.158	0.158	0.158	0.139	0.157
All Features	0.158	0.280	0.154	0.158	0.168	0.158	0.156	0.157	0.158	0.182	0.158	0.158	0.158	0.139	0.157
Regressor Aveg	0.158	0.276	0.156	0.158	0.168	0.158	0.156	0.156	0.158	0.180	0.158	0.158	0.158	0.139	0.158

(B)

	LarsCV	Lars	RandomForest (RF)	NuSVR	SVR	Ridge	GradientBoosting	AdaBoost	Bagging	DecisionTree	Lasso	BayesianRidge	ElasticNet	XGB	Polynomial
Counting	0.001	-0.005	-0.030	-0.002	-0.035	-0.003	0.008	0.006	-0.104	-0.973	0.001	0.001	0.001	-0.368	-0.001
POS Weighting	0.001	-0.047	-0.081	-0.009	-0.038	-0.007	0.002	0.006	-0.160	-0.916	0.001	0.001	0.001	-0.368	-0.001
TFIDF	0.001	0.005	0.008	-0.004	-0.036	0.000	0.015	0.016	-0.025	-1.098	0.001	0.001	0.001	-0.271	0.000
BoW Features	0.001	-85.47	0.007	0.001	-0.034	0.000	0.012	0.008	-0.081	-0.905	0.001	0.001	0.001	-0.369	-0.002
MED	0.001	0.000	-0.085	0.001	-0.035	0.000	0.012	0.010	-0.149	-0.886	0.001	0.001	0.001	-0.366	0.006
LCSubstring	0.001	0.001	-0.068	-0.003	-0.035	0.001	0.016	0.011	-0.130	-1.030	0.001	0.001	0.001	-0.366	0.005
LCSubsequence	0.001	0.002	-0.072	0.000	-0.035	0.002	0.019	0.018	-0.126	-0.941	0.001	0.002	0.001	-0.367	0.005
Lexical Features	0.001	-0.001	-0.039	0.000	-0.035	-0.001	0.000	0.014	-0.126	-1.084	0.001	0.001	0.001	-0.367	0.003
All Features	0.001	-3.059	0.001	0.001	-0.034	-0.008	0.004	0.001	-0.091	-1.003	0.001	0.001	0.001	-0.369	-0.010
Aveg	0.001	-9.842	-0.040	-0.002	-0.035	-0.002	0.010	0.010	-0.110	-0.982	0.001	0.001	0.001	-0.357	0.001

(C)

Figure 8: Results of different regressors on the BOW and lexical-based features according to (A) RMSE evaluation metric, (B) MAE evaluation metric, and (C) R^2 evaluation metric

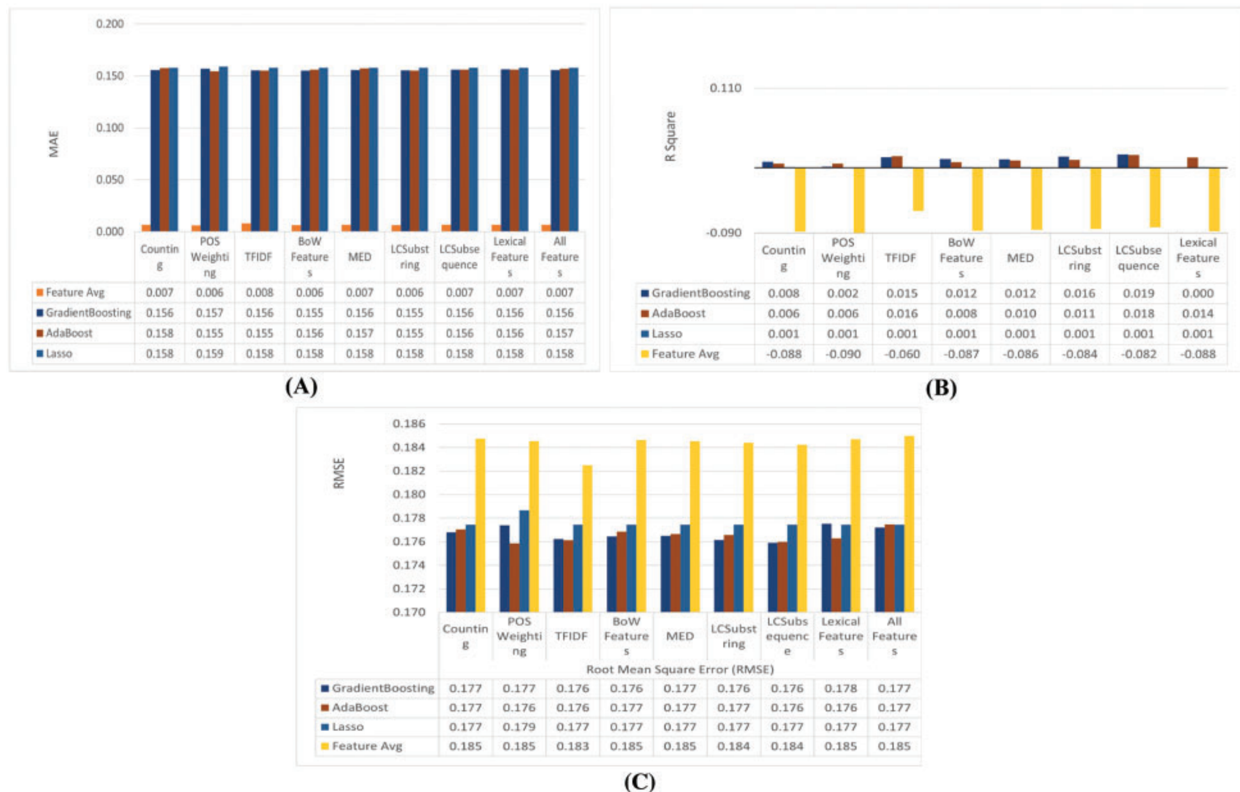


Figure 9: Results of the best regressors on the BOW and lexical-based features according to (A) MAE evaluation metric, (B) R^2 evaluation metric, and (C) RMSE evaluation metric

5.3 Pre-Trained Models Experiment

In this experiment, we aim to filter the best regressors according to each sentence embedding (averaging word vectors, projected averaging word vectors, and SIF) to select the best pool of regressors and the best representation of sentence embeddings using the same evaluation metrics settings as shown in Fig. 10. According to Fig. 11, those regressors (AdaBoost, Gradient Boosting, and Ridge) show the best results. Hence, we use them to identify the best sentence embedding representations. We conclude that the best sentence embedding representation is SIF sentence embeddings.

5.4 AraBERT as a Features-Extracting Model Experiment

In this experiment, we aim to filter which are the best regressors according to (AraBERT v0.1, AraBERT v1, AraBERT v0.2, AraBERT v2, and mBERT) embedding models to select the best pool of regressors and the best of these embedding models with the same evaluation metrics settings as the previous, as shown in Fig. 12. According to Fig. 12, those four regressors (AdaBoost, Gradient Boosting, Elastic Net, and Lars) show the best results. Thus, we use them to identify the best embedding models. As shown in Fig. 13, we conclude that the best embedding models are the AraBERT v2-large embedding model and the AraBERT v0.2-large embedding model.

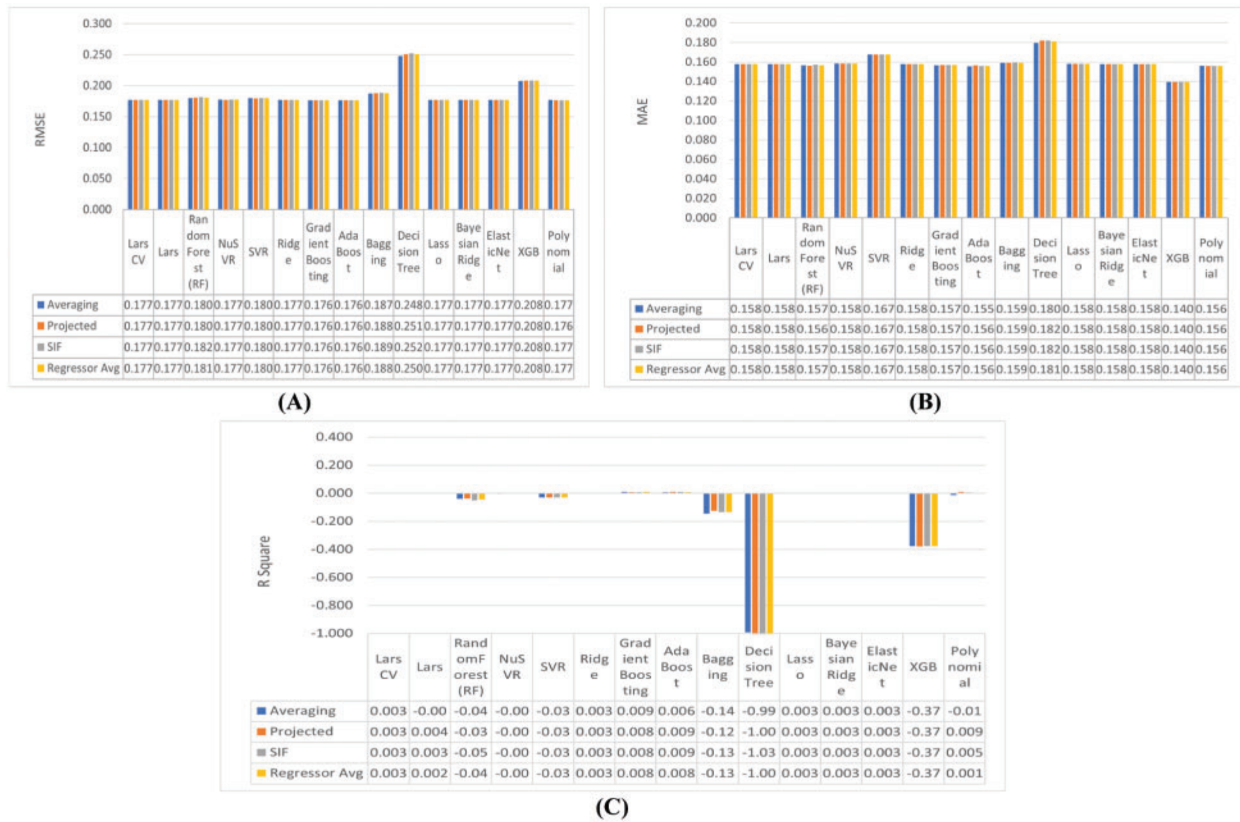
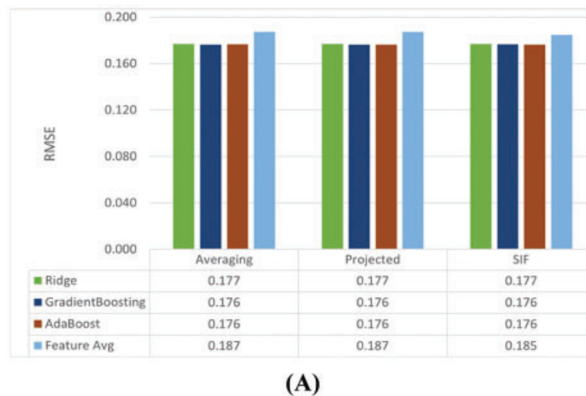


Figure 10: Results of different regressors on the sentence embeddings representations according to (A) RMSE evaluation metric, (B) MAE evaluation metric, and (C) R^2 evaluation metric



(A)

Figure 11: (Continued)

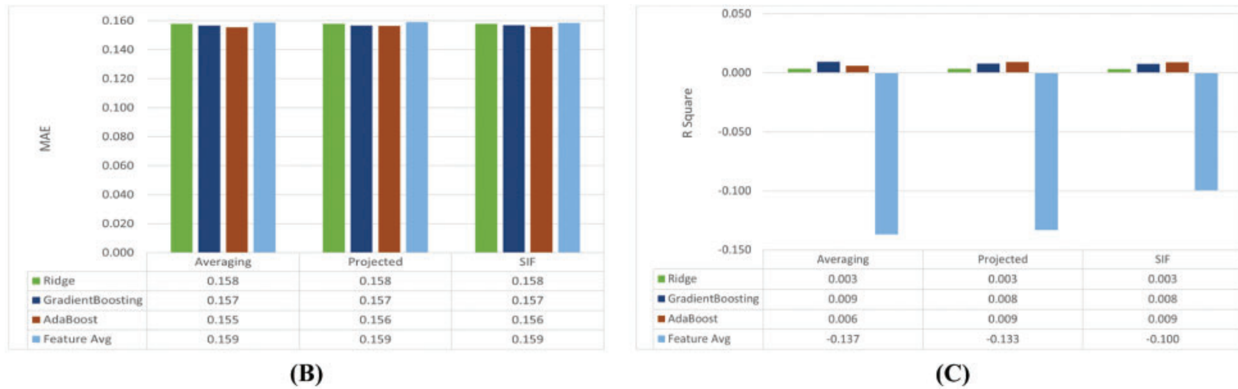


Figure 11: Results of the best regressors on the sentence embeddings representations according to (A) RMSE evaluation metric, (B) MAE evaluation metric, and (C) R^2 evaluation metric

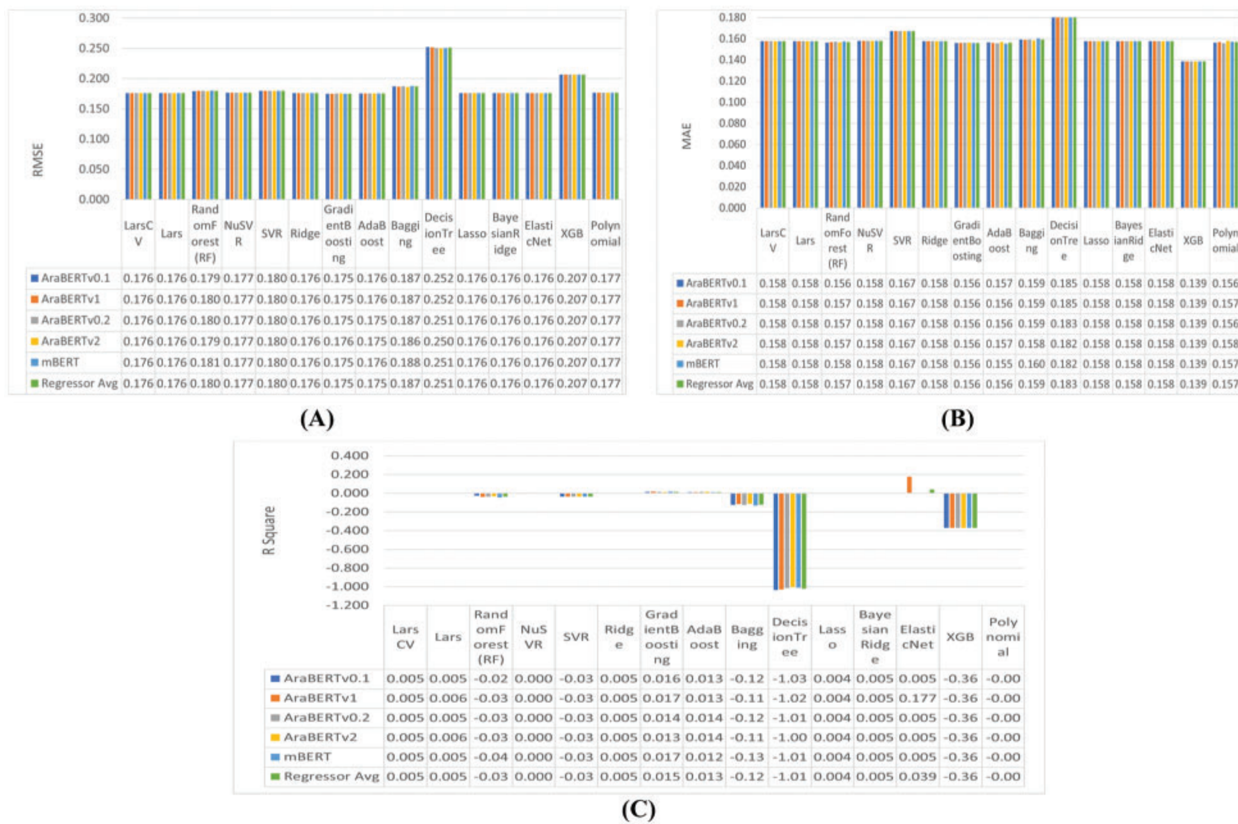


Figure 12: Results of different regressors on the five types of features according to (A) RMSE evaluation metric, (B) MAE evaluation metric, and (C) R^2 evaluation metric

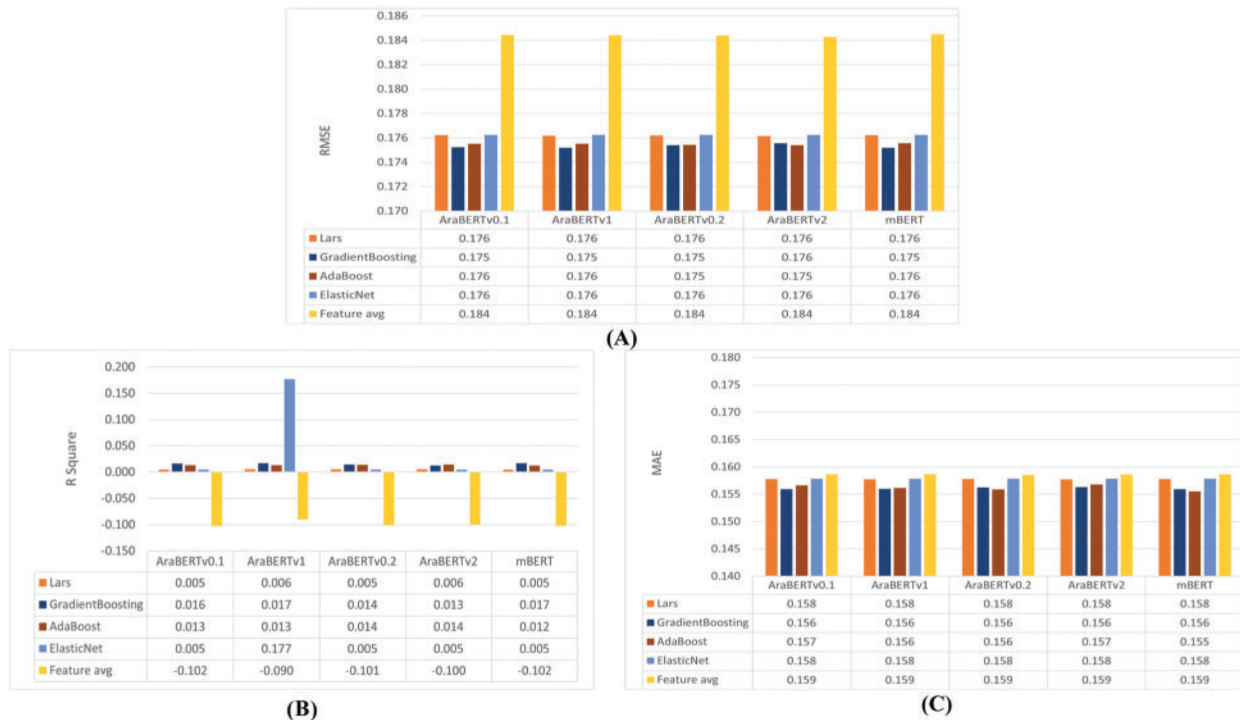


Figure 13: Results of the best regressors on the five types of features according to (A) RMSE evaluation metric, (B) $\boldsymbol{R}^2$ evaluation metric, and (C) MAE evaluation metric

6 Discussion of Results and Implications

In this paper, we used the AraBERT model in two different variants to estimate the similarity scores between text units. All the previous experiments demonstrate that these regressors (Gradient Boosting and AdaBoost) give the best results with these text embeddings (Long Common Substring, Long Common Subsequence, SIF, AraBERT v0.2, and AraBERT v2) in terms of RMSE, MAE, and R^2. We compare the findings of the MLP model in terms of MAE and MAPE with Gradient Boosting and AdaBoost regressors on these embeddings, as shown in Tabs. 6 and 7. In both previous tables, the bolded values in each row represent the best values (the smallest values) according to MAE and MAPE metrics, on the condition that the value of the metric for these values is positive. Consequently, some rows in Tab. 7 contain two bolded values. For example, the best MAPE to SIF value is 21.1508 and the worst value is −0.0065. Consequently, the condition is not met, and 21.7922 is chosen. The same is true for the BOW features. According to MAE and MAPE, BOW features are eliminated because the best regressor for them varies. From both tables, we can determine the best regressor for each embedding model as follows: AraBERT v0.2-Large with AdaBoost, Long Common Substring with AdaBoost, SIF with AdaBoost, Long Common Subsequence with MLP, and AraBERT v2-Large with GradientBoosting.

In the final experiment, we fine-tune the parameters of AraBERT v2 on the used dataset to estimate the relevancy scores between text units. Next, the comparison between the previous candidate models from Tab. 7 and the fine-tuned AraBERT v2 is illustrated in Tab. 8 in terms of the MAPE metric. Finally, we conclude that AraBERT v0.2-Large as a feature extractor model with AdaBoost has

the highest value in terms of R^2 and the variance in the MAPE values between it and others is minor. In addition, the AraBERT v0.2-Large as a feature extractor outperforms the fine-tuned AraBERT v2 model on the used data set in terms of R^2.

Table 6: Comparison between the MLP, gradient boosting, and Adaboost regressors according to MAE evaluation metric

	Regressor name		
Embeddings models	Gradient boosting	AdaBoost	MLP
Long common substring	0.155441	**0.155090**	0.156000
Long common subsequence	0.155957	0.156038	**0.156000**
BOW features	**0.155132**	0.156042	0.158000
SIF	0.156926	**0.155889**	0.156000
AraBERT v0.2-large	0.156253	**0.155855**	0.157000
AraBERT v2-large	**0.156323**	0.156772	0.157000

Table 7: Comparison between the MLP, gradient boosting, and Adaboost regressors according to MAPE evaluation metric

	Regressor name		
Embeddings model	Gradient boosting	AdaBoost	MLP
Long common substring	22.1845	**21.7857**	21.8181
Long common subsequence	22.2269	22.2616	**21.7953**
BOW features	22.1519	22.1232	**22.0282**
SIF	21.8898	21.7922	**21.1508**
AraBERT v0.2-large	21.9244	21.7723	21.9111
AraBERT v2-large	**21.9086**	21.9831	22.0359

Table 8: Best regressor for each embedding model according to our experiments

Embeddings model	MAPE	$\mathbf{R^2}$
Long common substring_AdaBoost	21.7857	0.011123
Long common subsequence_MLP	21.7953	0.000422
SIF_AdaBoost	21.7922	0.008748
AraBERT v0.2-large_AdaBoost	**21.7723**	**0.014050**
AraBERT v2-large_GradientBoosting	21.9086	0.012529
Fine-tuned AraBERTv2	21.8211	−0.032861

According to our findings in this paper, we can use them as the first step in a variety of NLP tasks such as text ranking, question–answer systems, and essay grading. Additionally, we used the multithreading concurrency concept to reduce processing execution time and increase CPU utilization as much as possible. Based on the findings, we believe that it is a better method for preprocessing text pairs than sequential processing and that it can be applied to other datasets and language settings.

7 Conclusions and Future Work

In this paper, we addressed the textual similarity task, which is of paramount importance for multiple topics in NLP, such as text ranking, essay grading, question answering systems, and text classification. We used the multi-tasks learning approach to train an algorithm to learn the embeddings from our dataset to estimate the textual similarity scores between text units and use them later in multiple tasks. Our system is divided into two different variants. In the first, we used multiple text vectorization schemes such as word counts, TFIDF, and POS weighting as statistical-based approaches, and FastText and Aravec pre-trained models as prediction-based approaches, besides the AraBERT as a feature extractor model to obtain text embeddings. These embeddings are then fed to various regressors to estimate the relevancy scores between text units. In the second variant of the system, we exploited the AraBERT model as a pre-trained model and fine-tuned its parameters for the task of measuring textual similarity. We conducted several experiments on the SemEval2017-task3-subtask-D dataset, and we proved that the usage of the AraBERT v0.2-Large as a feature extractor model with AdaBoost has the highest value in terms of R^2. In addition, the variance in the MAPE values between it and the other models is minor. Moreover, we noticed that the usage of AraBERT v0.2-Large as a feature extractor outperforms the fine-tuned AraBERT v2 model on the used data set in terms of R^2. As for future work, we intend to use the obtained similarity scores in other NLP tasks and use the AraGPT or AraELECTRA models to obtain different embeddings.

Acknowledgement: This paper and the research behind it would not have been possible without the exceptional support of my God, my supervisors, my family, and my institution and colleagues.

Funding Statement: The authors received no specific funding for this study.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] A. Abo-Elghit, A. Al-Zoghby and T. Hamza, "Textual similarity measurement approaches: A survey (1)," *The Egyptian Journal of Language Engineering*, vol. 7, no. 2, pp. 41–62, 2020. http://dx.doi.org/10.21608/ejle.2020.42018.1012.

[2] W. H. Gomaa and A. A. Fahmy, "A survey of text similarity approaches," *International Journal of Computer Applications(IJCA)*, vol. 68, no. 13, pp. 13–18, 2013. http://dx.doi.org/10.5120/11638-7118.

[3] M. A. Zahran, A. Magooda, A. Y. Mahgoub, H. Raafat, M. Rashwan *et al.*, "Word representations in vector space and their applications for arabic," in *Int. Conf. on Intelligent Text Processing and Computational Linguistics, CICLing 2015. Proc.: Lecture Notes in Computer Science (LNCS 9041)*, Cairo City, Egypt, pp. 430–443, 2015.

[4] J. Brownlee, *Deep Learning with Python: Develop Deep Learning Models on Theano and Tensorow Using Keras*, 1st ed. Victoria, Vermont, Australia: Machine Learning Mastery, 2016. [Online]. Available: https://www.goodreads.com/en/book/show/34043770-deep-learning-with-python.

[5] C. Lioma and R. Blanco, "Part of speech based term weighting for information retrieval," in *30th Annual European Conf. on Information Retrieval Research (ECIR 2009)*, Toulouse, France, pp. 412–423, 2009.

[6] D. Jurafsky and J. H. Martin, "N-Gram language models N-Gram language models," in *Speech and Language Processing*, 2nd ed., Upper Saddle River, NJ, USA: Prentice-Hall, Inc., vol. 3, pp. 189–230, 2009.

[7] T. Mikolov, I. Sutskever, K. Chen, G. Corrado and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. of the 26th Int. Conf. on Neural Information Processing Systems(NIPS 2013)*, vol. 26, Lake Tahoe, Nevada, USA, pp. 1–9, 2013.

[8] Q. Le and T. Mikolov "Distributed representations of sentences and documents," in *31st Int. Conf. on Machine Learning (ICML 2014)*, vol. 4, Beijing, China, pp. 2931–2939, 2014.

[9] E. Grave, P. Bojanowski, P. Gupta, A. Joulin and T. Mikolov, "Learning word vectors for 157 languages," in *Int. Conf. on Language Resources and Evaluation (LREC 2018)*, 11th ed., Miyazaki, Japan, pp. 3483–3487, 2019.

[10] J. Pennington and R. Socher, "GloVe: Global vectors for word representation," in *Proc. of the 2014 Conf. on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, Qatar, pp. 1532–1543, 2014.

[11] A. B. Soliman, K. Eissa and S. R. El-Beltagy, "AraVec: A set of arabic word embedding models for use in arabic NLP," *Procedia Computer Science*, vol. 117, pp. 256–265, 2017. http://dx.doi.org/10.1016/j.procs.2017.10.117.

[12] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark *et al.,* "Deep contextualized word representations," in *Proc. of the 2018 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, New Orleans, Louisiana, pp. 2227–2237, 2018. http://dx.doi.org/10.18653/v1/n18-1202.

[13] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," in *ACL 2018-56th Annual Meeting of the Association for Computational Linguistics, Proc. of the Conf. (Long Papers)*, vol. 1, Melbourne, Australia, pp. 328–339, 2018. http://dx.doi.org/10.18653/v1/p18-1031.

[14] J. Devlin, M. W. Chang, K. Lee and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proc. of the 2019 Conf. of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, vol. 1, no. Mlm, Minneapolis, U.S., pp. 4171–4186, 2019.

[15] T. Pires, E. Schlinger and D. Garrette, "How multilingual is multilingual BERT?," in *ACL 2019-Proc. of the 57th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, Florence, Italy, pp. 4996–5001, 2020. http://dx.doi.org/10.18653/v1/p19-1493.

[16] W. Antoun, F. Baly and H. Hajj, "AraBERT: Transformer-based model for arabic language understanding," in *Proc. of the 12th Language Resources and Evaluation Conf. (LREC 2020 Workshop)*, Marseille, France, pp. 9–15, 2020.

[17] A. Aquino and E. Chavez, "Analysis on the use of Latent Semantic Indexing (LSI) for document classification and retrieval system of PNP files," in" *Open Access Proceedings in Materials Science, Engineering and Chemistry (MATEC) Web of Conferences*, vol. 189, Beijing, China, pp. 3009, 2018. http://dx.doi.org/10.1051/matecconf/201818903009.

[18] A. H. Osman and O. M. Barukub, "Graph-based text representation and matching: A review of the state of the art and future challenges," *IEEE Access*, vol. 8, pp. 87562–87583, 2020. http://dx.doi.org/10.1109/ACCESS.2020.2993191.

[19] P. Jafarzadeh and F. Ensan, "A semantic approach to post-retrieval query performance prediction," *Information Processing & Management*, vol. 59, no. 1, pp. 102746, 2022. http://dx.doi.org/10.1016/j.ipm.2021.102746.

[20] M. Pan, J. Wang, J. X. Huang, A. J. Huang, Q. Chen *et al.,* "A probabilistic framework for integrating sentence-level semantics via BERT into pseudo-relevance feedback," *Information Processing & Management*, vol. 59, no. 1, pp. 102734, 2022. http://dx.doi.org/10.1016/j.ipm.2021.102734.

[21] A. A. Aliane and H. Aliane, "Evaluating SIAMESE architecture neural models for Arabic textual similarity and plagiarism detection," in *4th Int. Symp. on Informatics and its Applications (ISIA)*, M'sila, Algeria, pp. 1–6, 2020.

[22] A. Youssef, M. Elattar and S. R. El-Beltagy, "A Multi-embeddings approach coupled with deep learning for arabic named entity recognition," in *2nd Novel Intelligent and Leading Emerging Sciences Conf. (NILES 2020)*, Giza, Egypt, pp. 456–460, 2020. http://dx.doi.org/10.1109/NILES50944.2020.9257975.

[23] F. El-Alami, S. El Alaoui and N. En-Nahnahi, "Contextual semantic embeddings based on fine-tuned AraBERT model for arabic text multi-class categorization," *Journal of King Saud University-Computer and Information Sciences*, 2021. http://dx.doi.org/10.1016/j.jksuci.2021.02.005.

[24] A. Altahhan, E. Atwell and A. N. Alsaleh, "Quranic verses semantic relatedness using AraBERT," in *Proc. of the Sixth Arabic Natural Language Processing Workshop*, Kyiv, Ukraine (Virtual), pp. 185–190, 2021.

[25] A. Wadhawan, "AraBERT and Farasa segmentation based approach for sarcasm and sentiment detection in Arabic tweets," in *Proc. of the Sixth Arabic Natural Language Processing Workshop*, Kyiv, Ukraine (Virtual), pp. 395–400, 2021.

[26] P. Nakov, L. Màrquez, W. Magdy, A. Moschitti, J. Glass *et al.,* "SemEval-2016 Task 3: Community question answering," in *Proc. of the 10th Int. Workshop on Semantic Evaluation (SemEval-2016)*, San Diego, California, pp. 525–545, 2016.

[27] T. Zerrouki, "Tashaphyne, Arabic light stemmer." 2012, [Online]. Available: https://pypi.python.org/pypi/Tashaphyne/0.2.

[28] K. Darwish and H. Mubarak, "Farasa: A new fast and accurate Arabic word segmenter," in *Proc. of the 10th Int. Conf. on Language Resources and Evaluation (LREC'16)*, Portorož, Slovenia, pp. 1070–1074, 2016.

[29] S. Bird, "NLTK: The natural language toolkit," in *Proc. of the ACL Interactive Poster and Demonstration Sessions*, vol. Proceeding, Barcelona, Spain: Association for Computational Linguistics, pp. 214–217, 2004.

[30] M. N. Al-Kabi, S. A. Kazakzeh, B. M. Abu Ata, S. A. Al-Rababah and I. M. Alsmadi, "A novel root based arabic stemmer," *Journal of King Saud University-Computer and Information Sciences*, vol. 27, no. 2, pp. 94–103, 2015. http://dx.doi.org/10.1016/j.jksuci.2014.04.001.

[31] A. Apostolico and C. Guerra, "The longest common subsequence problem revisited, " *Algorithmica*, vol. 2, no. 2, pp. 315–336, 1987. http://dx.doi.org/10.1007/BF01840365.

[32] A. F. Gad, "Implementing the levenshtein distance in python," *Paperspace Blog*, 2019. [Online]. Available: https://blog.paperspace.com/implementing-levenshtein-distance-word-autocomplete-autocorrect/ (accessed Oct. 22, 2021).

[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion *et al.,* "Feature extraction," in *Scikit-learn 1.0 Documentation*, 2011. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html (accessed Oct. 22, 2021).

[34] R. Rehurek and P. Sojka, "Software framework for topic modelling with large corpora," in *Proc. of the LREC 2010 Workshop on New Challenges for NLP Frameworks*, Valletta, Malta, pp. 45–50, May 2010.

[35] S. Arora, Y. Liang and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," in *Int. Conf. on Learning Representations, ICLR (2017)*, Palais des Congrès Neptune, Toulon, France, 2017.

[36] T. Chen and C. Guestrin, "XGBoost: A scalable tree boosting system," in *Proc. of the ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining*, vol. 13–17-Augu, New York, NY, United States, pp. 785–794, 2016. https://doi.org/10.1145/2939672.2939785

[37] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion *et al.,* "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[38] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd Int. Conf. on Learning Representations, ICLR 2015*, San Diego, CA, USA, pp. 1–15, 2015.

[39] S. G. Andreas and C. Müller, "Evaluation of regression models in scikit-learn," in *Introduction to Machine Learning with Python: A Guide for Data Scientists*, 1st ed., vol. 5, Gravenstein Highway North, Sebastopol, CA, USA: O'Reilly Media, 2016. [Online]. Available: https://www.goodreads.com/book/show/24346909-introduction-to-machine-learning-with-python.

[40] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau *et al.,* "Theano: A python framework for fast computation of mathematical expressions," *arXiv e-prints*, vol. abs/1605.0, May 2016, [Online]. Available: http://arxiv.org/abs/1605.02688.

[41] A. Malakhov, "Composable multi-threading for python libraries," in *Proc. of the 15th Python in Science Conf. (SciPy 2016)*, Austin, Texas, pp. 15–19, 2016. http://dx.doi.org/10.25080/Majora-629e541a-002.
[42] A. Astori, "Concurrency and parallelism in python," *Towards Data Science*, Apr. 2021. [Online]. Available: https://towardsdatascience.com/concurrency-and-parallelism-in-python-bbd7af8c6625 (accessed Oct. 22, 2021).
[43] S. Raschka, "An introduction to parallel programming using Python's multiprocessing module–using Python's multiprocessing module," *sebastianraschka.com*, Jun. 2014. [Online]. Available: https://sebastianraschka.com/Articles/2014_multiprocessing.html (accessed Junuary. 3, 2022).
[44] P. M. Swamidass, "MAPE (mean absolute percentage error)," in *Encyclopedia of Production and Manufacturing Management*, 6th ed., Boston, MA: Springer US, pp. 462, 2000.