Applied Natural Language Processing 2023 S1 Assignment 2: Building a text matching system for question matching

Akide Liu
STUDENT NO. 1743748

May 30, 2023

Report submitted for **COMP SCI 4817** at the Faculty of Sciences, Engineering and Technology, University of Adelaide

In this report, first-person pronouns such as "WE" or "I" have been used interchangeably to refer to the **author's own work**, in order to maintain a professional and academic tone.

1 Text Pre-processing and Data Preparation

1.1 Removal of Stop Words

Stop words, commonly found in a language and devoid of significant semantic or pragmatic meaning, often hold little value in tasks related to natural language processing, such as sentiment analysis [9]. Examples encompass common English words like ”a”, ”an”, ”the”, ”in”, ”on”, ”is”, ”and”, and ”of”.

Efficiently removing stop words from a text corpus is beneficial in numerous ways: it mitigates noise, enhances computational efficiency, improves the accuracy of subsequent analyses, and decreases memory and computational resource utilization. We achieved this by identifying and eliminating stop words within our text corpus using the predefined English stop words set available in the Natural Language Toolkit (NLTK). The results are documented in the accompanying notebook, showing a noteworthy decrease in the token count from 900 to 580 within a singular text record.

The stop words removal operation was performed on a Pandas DataFrame, primarily utilizing a CPU. To bolster efficiency, we integrated a parallel execution mechanism, harnessing multiple threads found in modern computing systems. We set the number of threads to 10, realizing a linear acceleration effect. Notably, our parallelization scheme expedited the processing time approximately 10 times given our settings.

1.2 Stemming

Stemming, a technique in natural language processing, minimizes a word to its root form or stem [7]. It simplifies text data analysis by consolidating inflected or derived words into a shared base form. For instance, ”run”, ”running”, and ”runner” all stem to ”run”.

Integrating stemming into natural language processing tasks has several benefits. It decreases vocabulary size and thus minimizes memory and computational requirements for subsequent analysis. Moreover, stemming can enhance the accuracy of analyses by reducing the distinct words under consideration. It also proves beneficial in information retrieval tasks, yielding identical results for documents containing variations of a word.

In our study, we utilized the Porter stemming algorithm offered by the Natural Language Toolkit (NLTK) [2] to stem words in our text corpus. The stemming process results are presented in the accompanying notebook, where we observed a notable reduction in token count due to the consolidation of inflected or derived words into their base forms.

1.3 Dataset Splitting and Preparation

Our dataset includes multiple fields: *id*, *qid1*, *qid2*, *question1*, *question2*, and *is_duplicate*. The dataset contains 363,192 entries in total, which were ingested into a pandas DataFrame for further processing. We constructed our Query set from the *question1* field, selecting the first 100 entries where $is\_duplicate = 1$. The original order of these entries was preserved to create our ground truth (GT), facilitating the identification of similar questions as marked by the dataset. Concurrently, we assembled the training set from the first 10,000 unique entries from the *question2* field. This approach not only optimized computational costs but also expedited algorithm development. For example, traversing the entire dataset (approximately 300,000 samples) took around 30 minutes, whereas traversing the reduced subset took just a few minutes, thus demonstrating the method’s efficacy.

2 Sentence Representation

The process of sentence representation is a critical component in the realm of text matching. The prevailing methodologies can be categorized broadly into two groups: word-level and sentence-level techniques. Our research is concentrated on the implementation of three primary strategies, each offering unique advantages in their applicability and efficacy.

Initially, we concentrate on word-based models, focusing predominantly on the Term Frequency - Inverse Document Frequency (TF-IDF) algorithm [3] . This computational model gauges the frequency of words within a document, contrasting it against the overall frequency within a corpus. The TF-IDF algorithm excels in determining the significance of words within a text, thus facilitating the process of matching relevant queries within a dataset based on their corresponding TF-IDF scores.

We subsequently delve into the application of pre-trained word embeddings, employing models such as the Global Vectors for Word Representation (GloVe) [10] . These models furnish rich and contextual representations of words, paving the way for constructing sentence embeddings. The development of these embeddings involves calculating average embeddings, which are then utilized to compute the cosine similarity between the query and the target. Moreover, we examine advanced embedding aggregation methods, inclusive of the frequency down-weight average [5] and first principle components removal techniques [1] .

Finally, we investigate sentence-based models, such as Sentence-BERT (SBERT) [11], engineered specifically to compute sentence-level representations. These models excel at capturing the semantic similarity of sentences, considering the overarching meaning and context that extends beyond individual words. The application of these models significantly enhances our capacity to understand and compare text on a multi-dimensional level.

2.1 TF-IDF

Term Frequency-Inverse Document Frequency (TF-IDF) is extensively used in information retrieval to highlight the significance of a word in a specific document compared to a larger corpus. TF-IDF excels at emphasizing words that are particularly important within a document relative to the entire document set. This capability is invaluable in domains such as text mining, search engines, and user modeling, where the main objectives include identifying the most relevant documents for a query and understanding the main themes in a single document or a group of documents. The TF-IDF value increases proportionally with the word frequency in a document, but is adjusted based on the frequency of the word in the entire corpus. This delicate balance enables adjustments to account for the fact that certain words tend to occur more frequently overall.

2.1.1 Term Frequency

The term frequency, represented as tf(t,d), it's relative frequency of words in the document d. It effectively measures the occurrence of a term in a document, Algorithm can collect occurrence across the entire document, assigning a higher score to more frequently appearing terms. The computation of term frequency can be expressed as follows:

Algorithm 1 Term Frequency

```
1: procedure TermFrequency($t, d$)
2:     $f_{t,d} \leftarrow$ count of term t in document d
3:     $f_d \leftarrow$ total number of terms in document d
4:     $tf \leftarrow \frac{f_{t,d}}{f_d}$
5:     return $tf$
6: end procedure
```

$$\mathrm{tf}(t,d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}} \tag{1}$$

In this equation, $f_{t,d}$ represents the raw count of a term in a document, specifically, the number of times term t appears in document d. The denominator represents the total count of terms in document d, considering each occurrence of the same term individually. Consequently, this formula assigns a higher value to words that have a greater frequency of occurrence within a document.

2.1.2 Inverse Document Frequency

Algorithm 2 Inverse Document Frequency

```
1: procedure InverseDocumentFrequency($t, D$)
2:     $N \leftarrow$ total number of documents in corpus D
3:     $df_t \leftarrow$ number of documents in D where term t appears
4:     $idf \leftarrow \log \frac{N}{df_t}$
5:     return $idf$
6: end procedure
```

The inverse document frequency is utilized to measure the informativeness of a word by determining its prevalence across all documents. It essentially quantifies the significance of a term and assigns a higher score to less commonly occurring terms. The computation of inverse document frequency can be expressed as follows:

$$\mathrm{idf}(t,D) = \log \frac{N}{|\{d \in D : t \in d\}|} \tag{2}$$

In this equation, N represents the total number of documents in the corpus ($N = |D|$), while $|\{d \in D : t \in d\}|$ denotes the count of documents in which the term t appears (i.e., $\mathrm{tf}(t,d) \neq 0$). It is worth noting that if a term is absent from the corpus, division by zero would occur. To avoid this, it is customary to adjust the denominator to $1 + |\{d \in D : t \in d\}|$.

2.1.3 TF-IDF

Finally, the TF-IDF score is computed as the product of term frequency and inverse document frequency:

$$\mathrm{tfidf}(t,d,D) = \mathrm{tf}(t,d) \cdot \mathrm{idf}(t,D) \tag{3}$$

Algorithm 3 TF-IDF

```
1: procedure TFIDF($t, d, D$)
2:     $tf \leftarrow$ TermFrequency($t, d$)
3:     $idf \leftarrow$ InverseDocumentFrequency($t, D$)
4:     $tfidf \leftarrow tf \cdot idf$
5:     return $tfidf$
6: end procedure
```

This formula amalgamates the two measures discussed earlier. The TF-IDF score for a word in a document will be high if the term frequently appears in that document (high term frequency) and is scarcely found in the corpus (high inverse document frequency). This makes TF-IDF an extremely effective measure for detecting important and relevant words in a document.

2.2 Word Embedding

Embeddings, numerical representations, find application in a variety of formats such as word, text, and document embeddings. These embeddings, computed from text inputs, encapsulate semantic relationships and create robust frameworks suitable for a myriad of applications, particularly in the context of question matching tasks. The utility of embeddings extends to fast retrieval, sorting, grouping, and other critical tasks in text analysis.

In our study, we utilized pre-trained Global Vectors for Word Representation (GloVe) to facilitate word embeddings. The construction of sentence embeddings was then executed based on these vector spaces, thus contributing to an efficient representation of semantic information.

2.2.1 Sentence embedding

Algorithm 4 Sentence embedding

```
1: procedure CalculateSentenceEmbedding(sentence)
2:     $tokens \leftarrow split(sentence)$
3:     $embeddings \leftarrow [word\_embeddings.get(token, np.ones((dim,)))\forall token \in tokens]$
4:     if $len(embeddings) == 0$ then
5:         return $np.zeros((dim,))$
6:     else
7:         return $np.mean(embeddings, axis = 0)$
8:     end if
9: end procedure
```

The algorithm presented is a procedure for calculating sentence embeddings. It transforms a sentence into a numerical representation by leveraging word embeddings.

- The sentence is split into individual tokens (typically words).
- For each token in the sentence, the algorithm attempts to fetch the corresponding word embedding. If a word does not have a pre-calculated embedding, the algorithm

assigns a default embedding (a vector of ones with the length equal to the dimension of the embeddings, denoted as ‘dim‘).

- If no embeddings are obtained (i.e., if the sentence did not contain any words that have corresponding embeddings), the algorithm returns a zero vector with a length equal to the embedding dimension (‘dim‘).
- If embeddings are obtained for some or all of the tokens in the sentence, the algorithm calculates the mean of these embeddings, resulting in a single vector that represents the entire sentence. This vector is then returned by the procedure.

2.2.2 Ranked Search

Algorithm 5 Search

```
1:  procedure SEARCH(query, top_k)
2:      $query \leftarrow preprocess(query)$
3:      $query\_embedding \leftarrow calculate\_sentence\_embedding(query)$
4:      $document\_norms \leftarrow norm(document\_embeddings, axis = 1)$
5:      $query\_norm \leftarrow norm(query\_embedding)$
6:      $cosine\_similarities \leftarrow np.dot(document\_embeddings, query\_embedding)/(document\_norms * query\_norm + 1e - 8)$
7:      $top\_k\_indices \leftarrow cosine\_similarities.argsort()[-top\_k :][:: -1]$
8:      $results \leftarrow [(documents[i], cosine\_similarities[i], i) \forall i \in top\_k\_indices]$
9:      return $results$
10: end procedure
```

This algorithm describes a search procedure that takes a query and returns the top k most relevant documents.

- Preprocessing: The input query is preprocessed, which typically involves normalization and tokenization operations.
- Calculate document norms: This step calculates the norm (or length) of the vector representation of each document in the corpus, which is later used for calculating cosine similarities. The axis=1 parameter signifies that the norm is calculated across each row (i.e., each document).
- Calculate cosine similarities: We calculate the cosine similarity by using the matrix product between he document embedding and query embedding, followed by divide multiplication of these embedding, we have further add a extremely small constant to avoid zero division.
- The final outcome includes the top k matched sentence, along with their corresponding cosine similarity scores and indices.

3 Evaluation

3.1 Evaluation Metrics

We utilized the top 2 and top 5 accuracy metrics to measure the match accuracy of different algorithms. These metrics assess the algorithm's ability to rank the true label within the topmost predictions, thereby measuring its effectiveness in retrieving relevant documents. The indices returned by the algorithm correspond to the location of the document in the original dataset. These indices are compared against the ground truth (GT) labels for the calculation of Top-K accuracy.

3.2 Evaluation Results

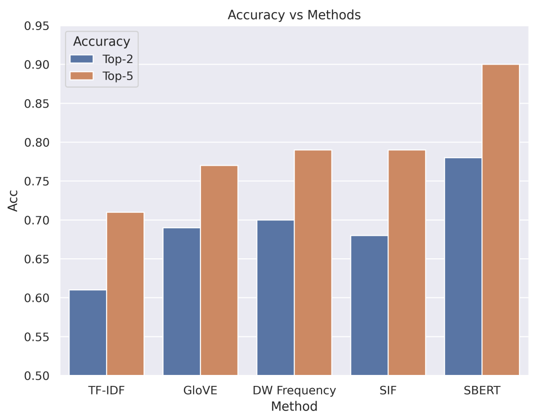


Figure 1: Accuracy vs Methods.

	Top-2	Top-5
TF-IDF	0.61	0.71
GloVE	0.69	0.77
DW Frequency	0.7	0.79
SIF	0.68	0.79
SBERT	**0.78**	**0.9**

Figure 2: Accuracy vs Methods.

In this section, we assess the performance of various matching algorithms, specifically evaluating Top-2 Accuracy and Top-5 Accuracy. The methodologies compared include TF-IDF, GloVE average sentence embeddings, sentence embeddings down-weighted by frequency, Smooth Inverse Frequency (SIF) based sentence embeddings, and Sentence-BERT (SBERT) with zero-shot learning.

As Table 2 clearly illustrates, sentence embedding-based methods surpass TF-IDF based approaches. This result underscores that sentence embeddings, which rely on large datasets for word embedding, can lead to more accurate matching. However, TF-IDF maintains practical value for time-sensitive tasks in industrial settings.

We further optimized the GloVE-based sentence embedding approach by down-weighting frequent words and removing first principles, resulting in enhanced performance and stability. Lastly, we appraised methods enhanced by deep learning. SBERT, also known as the sentence transformer, utilizes a dual-BERT architecture and targeted training. This approach yielded promising results, achieving 78% top-2 accuracy and 90% top-5 accuracy.

Algorithm 6 Calculate Sentence Embedding

```
1: procedure CALCULATE_SENTENCE_EMBEDDING(sentence)
2:     $tokens \leftarrow$ split($sentence$)
3:     Initialize $embeddings$, $weights$ as empty lists
4:     for token in tokens do
5:         if token in word_embeddings then
6:             $embeddings$.append($word\_embeddings[token]$)
7:             $weights$.append($\log \frac{a}{a+\text{word\_frequencies}[token]}$)
8:         else
9:             $embeddings$.append(np.ones(dim))
10:            $weights$.append($1e-8$)
11:        end if
12:    end for
13:    if $embeddings$ is not empty then
14:        $embedding \leftarrow$ np.average($embeddings, axis = 0, weights = weights$)
15:    else
16:        $embedding \leftarrow$ np.zeros(dim)
17:    end if
18:    return $embedding$
19: end procedure
```

4 Ablation Study

4.1 Down-weights frequency words

And here is the mathematical equation for down weighting frequency words used in the algorithm:

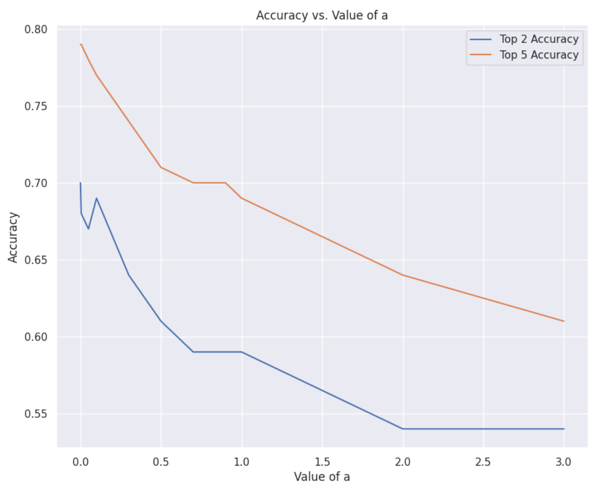
$$w_i = \log \frac{a}{a + f_i}$$

where w_i is the weight of the i-th word in the sentence, a is a parameter (usually a small number) to control the strength of the weighting, and f_i is the frequency of the i-th word. Down-weighting frequent words can improve the performance of language models for several reasons including Emphasizes Meaningful Words, reduces Noise, Addressing Bias, Improved Discrimination.

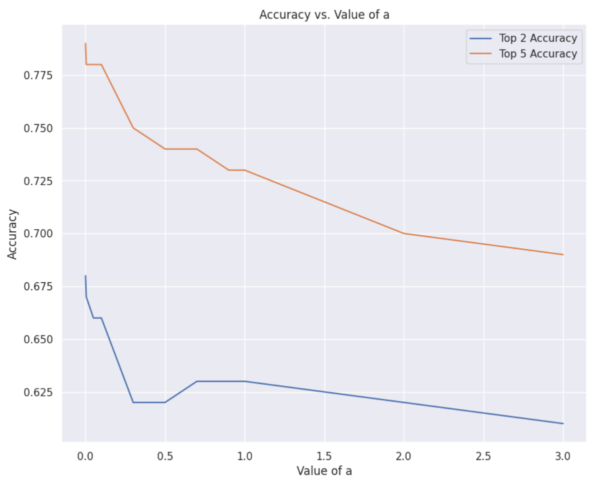
4.2 SIF Sentence Embedding

The method described in the algorithm from the paper "A Simple but Tough-to-Beat Baseline for Sentence Embeddings" by Arora et al [1] . provides a novel way to construct sentence embeddings by taking into account both the frequency and meaning of the words in the sentence. This methods has following benefits compared to the down-weighting frequency words directly while we can see that this methods provide less sensitive from the directly apply down-weighting frequency words in fig. 3 :

- Accounting for Semantics: This method uses word embeddings (such as those obtained from word2vec or GloVe), which capture semantic information about the words. Down-weighting frequent words alone would not account for the semantics of the words.



(a) w/o SIF



(b) w SIF

Figure 3: Evaluate the hyper-parameters of down-weight factor α with accuracy for different strategy.

Algorithm 7 SIF Sentence Embedding

```
Require: Word embeddings $\{v_w : w \in V\}$, a set of sentences $S$, parameter $a$ and estimated probabilities $\{p(w) : w \in V\}$ of the words.
Ensure: Sentence embeddings $\{v_s : s \in S\}$
1: for each sentence $s$ in $S$ do
2:     $v_s \leftarrow \frac{1}{|s|} \sum_{w \in s} \frac{a}{a+p(w)} v_w$
3: end for
4: Form a matrix $X$ whose columns are $\{v_s : s \in S\}$, and let $u$ be its first singular vector
5: for each sentence $s$ in $S$ do
6:     $v_s \leftarrow v_s - uu^T v_s$
7: end for
```

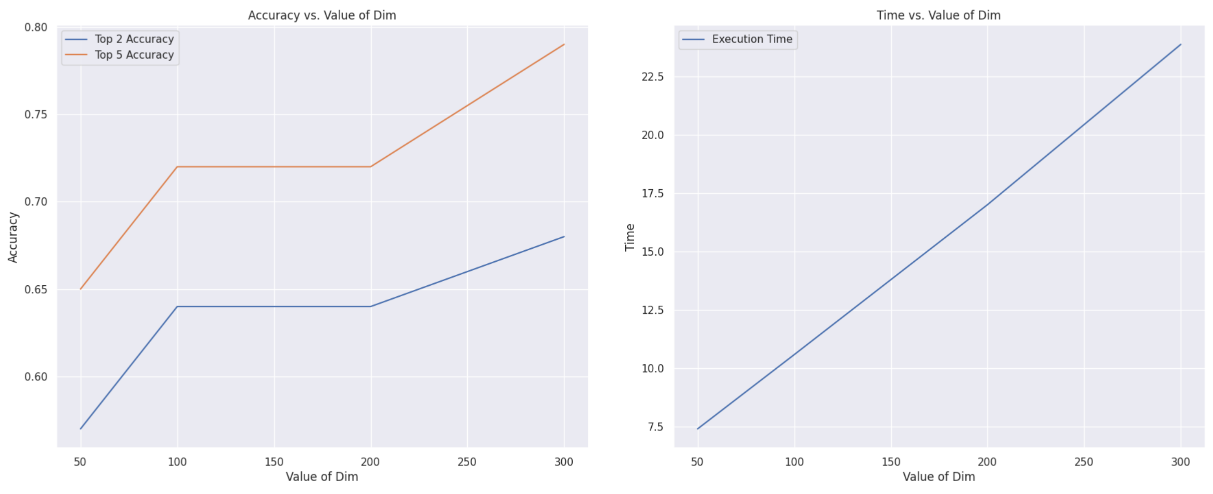
- Contextual Importance: The $\frac{a}{(a+p(w))}$ term in the weighting ensures that words that are important in the sentence context (lower p(w)) get more weight. This is more nuanced than just down-weighting based on overall frequency, as it takes into account the relative importance of a word in the specific context of a sentence.

- Removing Commonalities: By subtracting the first principal component, this method removes commonalities across the sentence vectors in the set. This helps eliminate components that might be related to common but less informative words (like stop words).

- Computationally Efficient: While this method is more complex than just down-weighting frequent words, it remains relatively computationally efficient and can be applied to large datasets.

4.3 Ablation of dimensionality of pre-trained word embedded.

In the fig. 4 , we found that with larger dimensional of the pre-trained GolVE word embeddings, the match system obtained higher accuracy while leads to higher computational



(a) Accuracy vs. Dimensionality (b) Computational Cost vs. Dimensionality

Figure 4: Accuracy and computational cost vs different dimensions for pre-trained word embeddings.

cost in terms of time. A interesting finding that the computational cost is growth linearly according to the dimension increased.

4.4 Sentence Transformer

Sentence Transformers represent a significant evolution in the field of natural language processing (NLP), providing an augmentation to Transformer-based models such as BERT [4] , DistilBERT [12] , and RoBERTa [8] by generating dense vector representations for sentences and paragraphs. Fine-tuning these models facilitates the closeness of semantically similar sentences in the vector space, enhancing the quality and speed of semantic similarity evaluations.

The role of Sentence Transformers is pivotal in tasks involving question matching, which entails the identification of semantic equivalence between two questions. This task presents complexities, as the identical query can be presented in various formats, using different wordings and structures. With their capacity to generate semantically relevant sentence embeddings, Sentence Transformers are capable of assessing the semantic similarity between two questions, thereby determining if they essentially solicit the same information. This capability is highly beneficial, as Sentence Transformers can comprehend the fundamental meaning and effectively discern the similarity, even when questions are articulated differently. This functionality has extensive applications in question-answering systems, chatbots, and information retrieval systems, to name a few.

In the present project, Sentence Transformers were utilized as zero-shot learners to execute zero-shot predictions on our target datasets. This methodology yielded encouraging results, registering a top-2 accuracy of 78% and a top-5 accuracy of 90%. Plans to fine-tune the Sentence Transformer using additional data over ten epochs resulted in an unanticipated decline in performance. Consequently, it was deemed necessary to employ the Sentence Transformers to induce overfitting on the training set. This involved the initial 10k questions in the dataset, which were identical to the evaluation sets. The performance continued to decline, suggesting that this pattern may be attributed to the

Table 1: Failure case analysis, give two example query and GT and 5 corresponding prediction and scores.

Query	GT	Prediction	Score
1. post question quora	post here	1. post question quora	1
		2. ask question quora	0.77648
		3. make question quora	0.70361
		4. post image quora	0.62506
		5. use quora	0.56807
2. trump s presidency affect indian students planning study us	bad trump s election president students aspiring study us	1. trump s presidency affect indian students planning study us	0.98994
		2. trump s victory affect india	0.75242
		3. impact donald trump s victory indian interests	0.74213
		4. donald trump s win mean indian students usa	0.73887
		5. trump s presidency affect indian industry	0.69425

model originally being trained on an extensive high-quality dataset, thereby equipping it with rich feature representations for generating sentence embeddings.

4.5 Failure case analysis & Discussion

The data quality of the dataset is considerably limited. Some failed cases were collected from the highest performing model, achieving 90% accuracy on the target dataset, as shown in Table 1. In the first question, the model outperformed the ground truth (GT) due to the first prediction being identical to the query. This reveals possible areas for improvement in the GT for achieving higher accuracy. For the second question, the query "trump s presidency affect indian students planning study us" is inferior to the GT "bad trump s election president students aspiring study us." According to our knowledge, the prediction is worse than the question.

Experiments demonstrate that further fine-tuning of pre-trained data on a smaller dataset with low-quality annotations could potentially impair the model's performance. This concept is supported by Chinchilla [6] , "Current large language models are 'significantly undertrained,' which is a consequence of blindly following the scaling hypothesis - making models larger isn't the only way toward improved performance." In light of this era of data-centric AI, it is increasingly evident that extensive and diverse datasets are fundamental requirements for the successful application of large language models (LLMs). Such limitations may prevent smaller research groups from achieving comparable results to those of larger corporations.

References

[1] Sanjeev Arora, Yingyu Liang, and Tengyu Ma. A simple but tough-to-beat baseline for sentence embeddings. In *International conference on learning representations*, 2017.

[2] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[3] Bijoyan Das and Sarit Chakraborty. An improved text sentiment classification model using tf-idf and next word negation. *arXiv preprint arXiv:1806.06407*, 2018.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[5] AbdAllah Elsaadawy, Marwan Torki, and Nagwa Ei-Makky. A text classifier using weighted average word embedding. In *2018 International Japan-Africa Conference on Electronics, Communications and Computations (JAC-ECC)*, pages 151–154. IEEE, 2018.

[6] Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

[7] Divya Khyani, BS Siddhartha, NM Niveditha, and BM Divya. An interpretation of lemmatization and stemming in natural language processing. *Journal of University of Shanghai for Science and Technology*, 22(10):350–357, 2021.

[8] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[9] Hans Peter Luhn. The automatic creation of literature abstracts. *IBM Journal of research and development*, 2(2):159–165, 1958.

[10] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[11] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*, 2019.

[12] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.