Noname manuscript No.
(will be inserted by the editor)

Supplementary File

Received: date / Accepted: date

A Keyword Extraction Techniques

Automatic keyword extraction is the task of identifying a small group of phrases, key phrases, keywords, or key segments from the unseen clusters that can define the intention or class of a particular cluster. Many text-based systems can be benefited since the keyword is the minor unit conveying the text or document's meaning. It is used in automated indexing, summary generation, categorization, clustering, filtering, topic detection and search, knowledge visualization, etc. Hence, keyword extraction can be viewed as the centre of all automatic document processing.

A.1 TF-IDF

A numerical statistic called the term frequency-inverse document frequency (TF-IDF) shows how vital a phrase is to a document in a cluster or corpus. Knowledge retrieval, text mining, and user modelling are frequently used as weighting characteristics. The TF-IDF significantly increases in direct ratio to the number of times a word arises in the document. TF-IDF is neutralized using several documents in the corpus that holds the word, which permits adjusting that some words are used more repeatedly than others.

The TF-IDF is the product $(\cdot)$ of term frequency (TF) and inverse document frequency (IDF). There are different methods for selecting the identical weights of both statistics. Term frequency, $T_{freq}\left(t_m, d_o\right)$ is the frequency of term t_m.

$$T_{freq}\left(t_m, d_o\right) = \frac{f_{t_m, d_o}}{\sum_{t'_m \in d_o} f_{t'_m}, d_o} \tag{1}$$

Where $f\left(t_m, d_o\right)$ is the number of times that term t_m appears in the document d_o. or a raw total of a term in a document.

Now IDF can be defined as ID_{freq} where

$$ID_{freq} = log \frac{N_c}{\{d_o \in D : t_m \in d_o\}} \tag{2}$$

Here N_c = is total number of documents, and $N_c = |D|$ with $d_o \in D$ number of documents where t_m appears, that is $t_m \in d_o : T_{freq}(t_m, d_0) \neq 0$). Now by equations 1 and

Address(es) of author(s) should be given

2 we can define TF-IDF as

$$T_{freq}ID_{freq}(t_m, d_o, D) = T_{freq}(t_m, d_o) \cdot ID_{freq}(t_m, D) \tag{3}$$

A.2 Yake

A standard unsupervised key-phrase extraction strategy is utilizing the TF-IDF. The candidate words' cases are reflected in the Word Casing (W_c). The Word Location (W_p) represents a word's position in the document; the more repeatedly a word arises in the front, the higher its value. The Term Frequency (W_f) metric indicates that the more frequently a word appears in a document, the higher its value. The Term Frequency (W_f) metric indicates that the more frequently a word appears in a document, the higher its value.

The amount of different words emerging on both sides of a candidate phrase is indicated by the Word Relatedness to Context (W_{RC}). The Word DifSentence (W_D) shows how often a candidate word appears in different sentences.

$$S(w) = \frac{W_R * W_p}{W} \tag{4}$$

A.3 TextRank Model for Keyword Extraction

Graph-based ranking algorithms determine the significance of a vertex within a graph based on global knowledge recursively drawn from the whole graph. The basic concept enforced by graph-based ranking techniques is recommendation or voting. When one vertex ties to another one, it votes for that different vertex. The more increased the number of votes cast for a vertex, the more elevated the significance of the vertex. Furthermore, the significance of the vertex casting a vote impacts the relevance of the vote itself, and the ranking model brings this knowledge into account. Therefore, the score associated with a vertex is specified. It is based on the votes cast and the score of the vertices releasing these votes.

The TextRank approach is fully unsupervised. First, the text is tokenized and tagged with an area of speech labels, which is a necessary pre-processing step before applying syntactic filters.

In order to discover relevant terms, the *TextRank* algorithm creates a word network. This network is built by examining which words are connected. If two words often appear next to each other in the text, a connection is created between them. The link would have a more significant weight if the two words appeared more frequently next to each other in the text. PageRank algorithm is applied to the above network, generating a keyword table.

A.4 SingleRank Model for Keyword Extraction

TextRank algorithm is the most famous graph-based ranking algorithm, the first to use PageRank for keyword extraction. TextRank deletes all function words, and only certain parts of speech (adjectives and names) can be candidate words. Then the algorithm connects all the selected candidate words and creates a directed unweighted graph. The final score of node (keyword) V_k is determined by Equation 5.

$$S(V_k) = (1 - ß) + ß \sum_{m \in NB(V_k)} \frac{1}{|NB(V_m)|} S(V_m) \tag{5}$$

ß refers to the damping factor to prevent it from recursive computations, and $NB(V_m)$ is the set of neighboring nodes of V_m.

On the other hand, SingleRank added weight based on TextRank between nodes appearing in window w simultaneously. The weight is calculated by the

number of times keywords occur together in the widow w simultaneously. The final score of nodes V_k for SingleRank is determined by Eq. 6.

$$S(V_k) = (1 - ß) + ß \sum_{m \in NB(V_k)} \frac{C_o(V_j, V_m)}{\sum_{j \in NB(V_m)} C_o(V_m, V_j)} S(V_m) \tag{6}$$

where $C_o(V_{i1}, V_{i2})$ represents the number of times that node V_{i1} & node V_{i2} appear together in a document. NB_k denotes the neighboring set of node k.

TextRank's networks are unweighted graphs. The weights of the edges might exhibit the strength of the semantic connection between the two nodes; utilizing the weighted graph in the key extraction job may be preferable. Wan and Xiao introduced SingleRank, based on this assumption, which added weights based on the TextRank between nodes emerging in the window of w at the same time, with the significant weight specified by the number of times the two words emerged in the window of w at the identical time.

A.5 KEA

KEA is an algorithm for automatic keyword extraction from the text; it functions in two phases. First, The training phase creates a model for determining keywords, employing the training documents where the keywords are known. Next, the extraction phase picks keywords from an unknown document using the above model. In both phases, pick many candidate words from their input documents and then estimate the weights of distinct features or attributes for each candidate.

The candidate words selection process follows three steps to execute candidate words. The first step processes the input text, recognizes candidates, and checks and case-folds the words. After breaking the text into phrases and sentences, KEA assumes all the sub-sequences in individual sentences and decides which appropriate candidate words. The second step estimates the feature and calculates two features for each candidate word employed in training and extraction. Then, it estimates a word's frequency in a document approximated to its aberration in general usage.

A.6 WINGNUS

WINGNUS analyzes the usage of logical structures (LS) to extract keywords. The concept is to determine the inquiry content in the candidate label technique while preserving scope. It pulls candidates according to the standard term rules. However, employing the entire document text as input shortens the input text at distinct levels from complete to minimal. To determine a candidate word, it uses different methods such as TF-IDF, word off-set (First and last occurrences), Altitude of terms in phrases, and available attributes in a document such as italic and bold letters or phrases that can be a hint of

the keyword. Apart from the above methods, it also uses conventional techniques such as the document's title, frequency of the title, and header/footer to indicate the frequency of a word.

A.7 KeyBERT

KeyBERT uses BERT-embedding to discover the sub-phases in a document that are the most similar to the document itself; it uses a cosine similarity. First, it creates a document-level representation using the BERT embedding. Then, extract word embedding for N-gram words/phrases and find the similarity among words/phrases that are the most identical to the document using the cosine similarity. These most identical words will represent the whole document.

B Sentence Embedding

B.1 Sentence BERT (SBERT) and different versions

SBERT extends the pre-trained Bidirectional Encoder Representations from Transformers (BERT) network (Devlin, Chang, Lee, & Toutanova, 2019) that employs triplet and Siamese network configurations to derive semantically significant sentence embedding that can be approximated utilizing cosine-similarity. Sentence BERT(SBERT) appends a pooling function to the outcome of conventional BERT to retrieve a fixed-sized sentence embedding. Sentence BERT(SBERT) appends a pooling function to the outcome of conventional BERT to retrieve a fixed-sized sentence embedding.

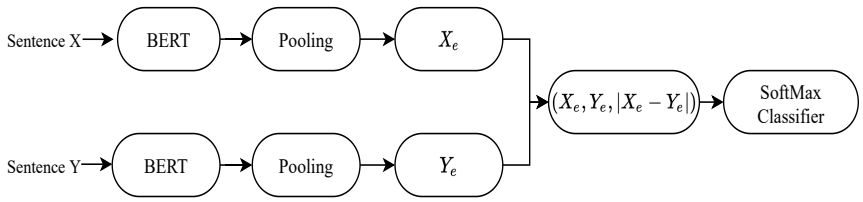

Fig. 1: SBERT framework with classification objective function where two different BERT networks have connected weights. Sentence embedding concatenate X_e and Y_e with the element-wise difference $|X_e - Y_e|$

The SBERT uses siamese and triplet networks to fine-tune BERT / Roberta. Siamese and triplet networks (Schroff, Kalenichenko, & Philbin, 2015) are used to adjust the weights so that the word embedding obtained is semantically relevant and can be compared using cosine similarity. It uses a classification objective function (see Figure 1) and an objective regression function (see Figure 2).

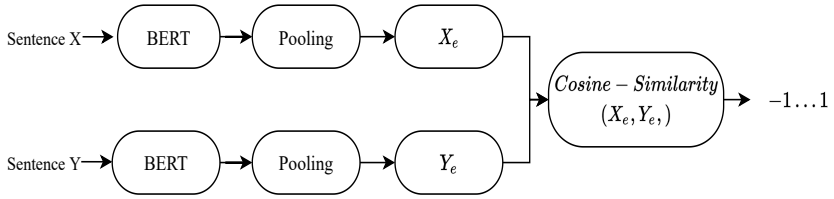

Fig. 2: SBERT framework to compute similarity scores. The objective regression function is used in this framework.

Sentence encoder *BaseBERT* is one of the strategies in the sentence BERT model in which it uses CLS-token, calculating the mean of all outcome vectors for pooling. *Paraphrase-distilroberta-base-v1* sentence- transformers model maps paragraphs to a dimensional dense vector space, size of 768, which can be used for semantic searches and clustering. Similarly, *sentence encoder Glove* uses average word embedding glove (Conneau, Kiela, Schwenk, Barrault, & Bordes, 2017) instead of BERT in SBERT.

B.2 Universal Sentence Encoder

The Universal Sentence Encoder (USE) is a publicly available encoding technique that encodes text into high dimensional vectors. These encoded sentences are used for text classification, clustering, natural language tasks, and semantic similarity searches. The USE has two interpretations, Deep Averaging Network (DAN) (Iyyer, Manjunatha, Boyd-Graber, & Daumé III, 2015) and Transformer encoder (Vaswani et al., 2017). Transformer encoder is computationally more intensive, but it has higher accuracy than Deep Averaging Network (DAN) encoding. However, the DAN has low accuracy, but it is computationally less expensive.

The encoding sub-graph of the transformer framework is employed to construct sentence embedding in the transformer-based sentence encoding model. This sub-graph employs attention to generate context-aware expressions of words in a phrase that accounts for sequencing and identifying all other words in the text. The element-wise aggregation of the expressions at apiece word location transforms the context-aware phrase representations into a set of length sentence encoding vectors. It takes the input of tokenized strings and provides a 512-dimensional vector as embedding.

This encoding model was created to be as versatile as possible that can be used for general purposes. This is achieved by employing multi-task learning, whereby a single encoding standard provides numerous downstream jobs. It supported Skip Thought-like tasks that belong to the unsupervised learning and ruining text, conversational data, and supervised training-based classification tasks. In a transformer, model LSTM replaces the Skip-Thought task.

C Performance Matrices

1. **Precision:** Precision indicates the percentage of prediction of positive instances among all positive instances. It is a proposition of instances projected as unseen that only actually are unseen.

$$P_{rec} = \frac{T_{p_o}}{T_{p_o} + F_{p_o}} \tag{7}$$

2. **Recall:** Also referred to as Sensitivity, It is the rate of true positive instances. The recall is a proportion of perceived positive instances forecast as positives concluded by the classifier.

$$R_{ec} = \frac{T_{p_o}}{T_{p_o} + F_{n_e}} \tag{8}$$

3. **Closed Accuracy:** It is the same as the classic accuracy score, where all the classes found in the test data are present in the training data. It is given by

$$ClAcc = \frac{\sum_{i=1}^{C} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}})}{\sum_{i=1}^{C} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}} + \mathrm{F_{p_o}} + \mathrm{F_{n_e}})} \tag{9}$$

4. **Open Accuracy:** The $ClAcc$ can be extended for OWML by extending the current accuracy evaluation matrices from C classes to C+1 classes, where C+1 is can be assumed as a class of unseen instance classes. The formula is given below.

$$OpAcc = \frac{\sum_{i=1}^{C+1} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}})}{\sum_{i=1}^{C+1} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}} + \mathrm{F_{p_o}} + \mathrm{F_{n_e}})} \tag{10}$$

5. **Binary Accuracy:** We consider this accuracy in evaluating how the proposed work performs when only identifying novel instances. A T_{p_o} is considered when the given instance is unseen, and the model also predicts the same. The formula is represented as:

$$BinAcc = \frac{T_{p_o} + T_{n_e}}{T_{p_o} + T_{n_e} + F_{p_o} + F_{n_e}} \tag{11}$$

6. **F1-score**: It provides a correctness measure to examine wrongly classified unseen instances. It is computed as the harmonic mean of P_{rec} and R_{ec}

$$F1\text{-}score = 2 \times \frac{P_{rec} \times R_{ec}}{P_{rec} + R_{ec}} \tag{12}$$

7. **MCC:** It finds the correlation coefficient between the actual class and the predicted class. The value of MCC varies between -1 and +1, where -1 represents the absolute disagreement between actual class values and predicted class values, and 1 indicates the absolute agreement between actual class values and predicted class values. It is calculated as:

$$MCC = \frac{(T_{p_o} \times T_{n_e}) - (F_{p_o} \times F_{n_e})}{\sqrt{(T_{p_o} + F_{p_o})(T_{p_o} + f_{n_e})(T_{n_e} + F_{p_o})(T_{n_e} + F_{n_e})}} \tag{13}$$

8. **G-mean:** The Geometric Mean (G-mean) is a statistic that assesses the balance between majority and minority categorization performance. Lower G-mean signifies poor performance in the classification of T_{p_o} cases even if T_{n_e} cases are correctly predicted. This measure prevents over-fitting of T_{p_o} instances and under-fitting of the T_{n_e} ones.
G-mean1: G-mean1 is the geometric mean of P_{rec} and R_{ec}.
G-mean2: G-mean2 is the geometric mean of S_{peci} and R_{ec}

$$GM1 = \sqrt{P_{rec} \times R_{ec}} \tag{14}$$

$$GM2 = \sqrt{R_{ec} \times S_{peci}} \tag{15}$$

9. **ARS:** It analyzes the similarity between all instances of two clusters by pair-wise comparison. It is computed as:

$$ARS = \frac{(RI - ExpectedRI)}{(max(RI) - ExpectedRI)} \tag{16}$$

Where RI is the rand index score; it is the similarity measure between two clusters by counting pairs of samples assigned in the same or different clusters in the predicted and actual clusters.

10. **NMI:** To label a class, we reduced the entropy of class labels using mutual information. For this, the NMI is computed as:

$$NMI(g_i, g_j'') = \sum_{i=1}^{|g_i|} \sum_{j=1}^{|g_j''|} \frac{|g_i \cap g_j''|}{N} \log \frac{N|g_i \cap g_j''|}{|g_i||g_j''|} \tag{17}$$

Where $|g_i|$ and $|g_j''|$ are number of instances in the cluster g_i and g_j'', respectively.

11. **FMS:** To analyze the clustering performance of the proposed work with ground truth, the FMS is calculated as:

$$FMS = \frac{T_{p_o}}{\sqrt{(T_{p_o} + F_{p_o}) * (T_{p_o} + F_{n_e})}} \tag{18}$$

D Ablation Study

The results for discovering unseen instances and identifying novel classes are reported in Table 5 and Table 6. We evaluated the proposed work with various techniques to validate the performance of three modules of the proposed work. We performed an ablation study to address the following three queries (1) Why does the proposed framework use Universal Sentence Encoder for text Embedding (Section D.0.1)? (2) How much BlendEmb enhances the performance of the proposed framework in terms of the instances similarity search (Section D.0.2)? (3) How many neighbors should be considered in neighborhood blending (Section D.0.3) ?

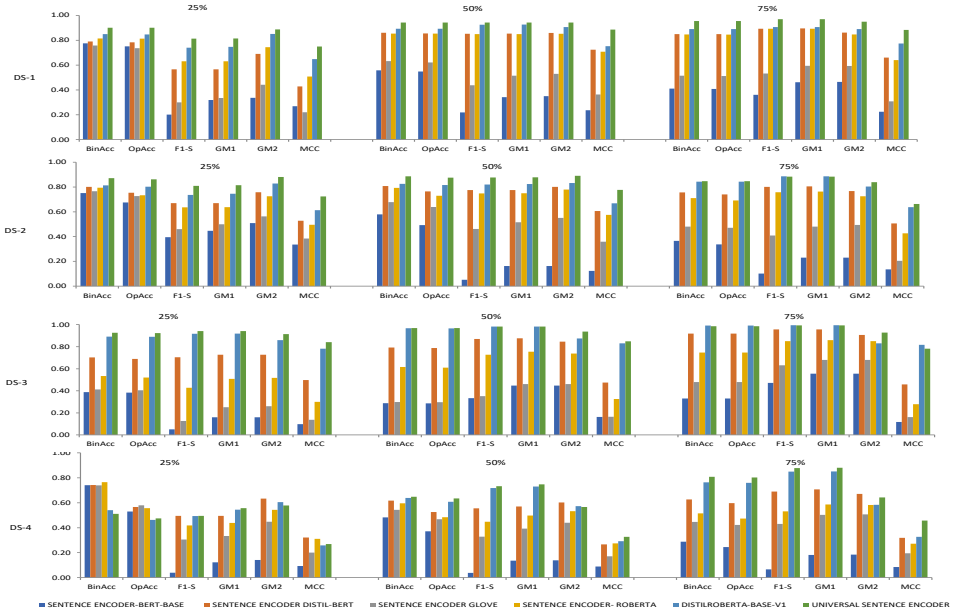

Fig. 3: The Proposed Framework's Performance with Different Embedding on DS-1, DS-2, DS-3, and DS-4

D.0.1 The Proposed Framework's Performance Analysis with Different Sentence Embedding

To address query (1), we compare the performance of the proposed work with different embedding techniques. We consider five embedding techniques Sentence-BERT (Reimers et al., 2019), Sentence Encoder Glove (Reimers et al., 2019), Paraphrase-distilroberta-base-v1 (Reimers et al., 2019), DistilBERT (Sanh, Debut, Chaumond, & Wolf, 2019), and Roberta (Liu et al., 2019) to analyze the performance. The performance is calculated with DS-1, DS-2, DS-3, and DS-4 for 25%, 50% openness, and 75%. The overall performance of the proposed work is better with a universal sentence encoder (Cer et al., 2018) compared to other existing embedding techniques. To analyze the performance of a universal sentence encoder, the proposed framework tested with five different embedding with the same experimental settings and performance metrics, i.e., Binary accuracy (BinAcc), Open accuracy (OpAcc), F1-Score (F1-S), Gmean-1 (GM1), Gmean-2 (GM2), and MCC. Figure 3 shows the comparative performance results with different embedding on DS-1, DS-2, DS-3, and DS-4. In the case of DS-1, Paraphrase-distilroberta-base-v1 has given the best result for 25%, 50%, and 75% openness except for a universal sentence encoder. The universal sentence encoder outperforms Paraphrase-distilroberta-base-v1. It enhances the performance by 5.17%, 5.50%, 7.40%, 6.85%, 3.62%, and 10.10% for BinAcc OpAcc, F1-S, GM1, GM2, and MCC, respectively. In terms of 50%

openness, it enhances performance by 5.00%, 5.00%, 1.75%, 1.64%, 3.70%, and 13.39%, for BinAcc, OpAcc, F1-S, GM1, GM2, and MCC, respectively. Similarly, for 75% openness, it enhances the performance by 6.50%, 6.50%, 6.29%, 6.28%, 6.04%, and 10.96% for BinAcc OpAcc, F1-S, GM1, GM2, and MCC, respectively.

In the case of DS-2, the universal sentence encoder outperforms existing embedding techniques. Paraphrase-distilroberta-base-v1 has given a better result after the universal sentence encoder. The universal sentence encoder enhances the performance of the proposed work by 5.86%, 5.86%, 7.29%, 6.90%, 5.33%, and 11.02% for BinAcc OpAcc, F1-S, GM1, GM2, and MCC, respectively. In the case of 50% openness, it improves performance by 6.00%, 6.00%, 5.70%, 5.44%, 5.81%, and 10.89% for BinAcc OpAcc, F1-S, GM1, GM2, and MCC, respectively. Similarly, in the case of 75% openness, it improves the performance by 0.43% and 0.43% for BinAcc and OpAcc, respectively. The F1-S and GM1 negligibly decline, but GM2 and MCC improve by 3.53% and 2.65%.

In the case of DS-3, The performance of the universal sentence encoder slightly declines with 25% openness for BinAcc and GM2. However, it improves the OpAcc, F1-S, GM1, and MCC by 1.14%, 0.13%, and 1.24%, respectively. With 50% openness, it improves performance by 0.96%, 2.63%, 1.43%, 1.43%, and 3.48% for BinAcc OpAcc, F1-S, GM1, and MCC, respectively. Similarly, 75% openness improves the performance by 4.43%, 4.30%, 2.91%, 2.95%, 6.01%, and 13.01% for BinAcc OpAcc, F1-S, GM1, GM, and MCC, respectively.

Table 1: Comparison of the proposed framework's performance without neighborhood blending and with neighborhood blending

	25%						50%						75%					
CLINC150 (DS-1)																		
	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC
#1	0.855	0.855	0.739	0.742	0.843	0.646	0.923	0.923	0.923	0.923	0.923	0.847	0.918	0.918	0.945	0.945	0.910	0.792
#2	0.902	0.902	0.814	0.815	0.887	0.749	0.943	0.943	0.943	0.943	0.943	0.887	0.955	0.955	0.970	0.970	0.950	0.883
SNIPS (DS-2)																		
#1	0.774	0.763	0.712	0.731	0.810	0.582	0.806	0.794	0.809	0.818	0.813	0.649	0.823	0.823	0.875	0.876	0.756	0.584
#2	0.871	0.861	0.809	0.814	0.881	0.723	0.886	0.876	0.877	0.878	0.891	0.777	0.847	0.847	0.883	0.884	0.838	0.664
GOOGLE SNIPPETS (DS-3)																		
#1	0.429	0.409	0.464	0.539	0.481	0.207	0.609	0.602	0.719	0.741	0.468	0.263	0.781	0.777	0.866	0.870	0.516	0.346
#2	0.511	0.475	0.495	0.557	0.579	0.270	0.648	0.635	0.733	0.748	0.566	0.327	0.809	0.803	0.879	0.881	0.643	0.457
DBPEDIA (DS-4)																		
#1	0.833	0.830	0.876	0.882	0.770	0.665	0.949	0.947	0.972	0.972	0.784	0.719	0.986	0.986	0.993	0.993	0.773	0.697
#2	0.926	0.923	0.942	0.942	0.914	0.841	0.970	0.969	0.983	0.983	0.936	0.849	0.986	0.986	0.992	0.993	0.927	0.782

Abbreviations: #1: Without *Blend_Emb*, #2: With *Blend_Emb*

D.0.2 The proposed framework's Performance Analysis with and without Blend_Emb

To address query (2), the proposed work is evaluated with and without neighborhood blending (*Blend_Emb*). We found that *Blend_Emb* enhances the performance of the proposed framework. Table 1 shows the performance analysis. The overall performance of the proposed work is enhanced with *Blend_Emb*,

except the F1-S for DS-4 with 75% openness. We have gained significantly remarkable improvements with 25% openness, and it also improves performance with 50% and 75% openness. Here we discuss the percentage of improvements with 25% openness.

In the case of DS-1, it enhances the BinAcc and OpenAcc 4.67%, F1-S 7.51%, GM1 7.27%, GM2 4.43%, and MCC 10.35% with 25% openness as compared to without *Blend_Emb*. Similarly, in the case of DS-2, it enhances the BinAcc 9.71%, OpenAcc 9.86%, F1-S 9.76%, GM1 8.26%, GM2 7.09%, and MCC 14.11%. In the case of DS-3, it enhances the BinAcc 8.20%, OpenAcc 6.58%, F1-S 3.06%, GM1 1.78%, GM2 9.81%, and MCC 6.23%. In the case of DS-4, it enhances the BinAcc 9.35%, OpenAcc 9.30%, F1-S 6.55%, GM1 6.02%, GM2 14.47%, and MCC 17.59%. Similarly, *Blend_Emb* improves performance for 50% and 75% openness for all the datasets.

D.0.3 The Proposed Framework's Performance Analysis with Different Set of Neighbors in neighborhood blending

Table 2: the proposed framework's performance with the k= 10, 15, and 20 most similar encoded sentences

	25%						50%						75%					
CLINC150 (DS-1)																		
	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC
K=10	0.887	0.887	0.783	0.784	0.863	0.708	0.932	0.932	0.930	0.931	0.931	0.864	0.937	0.937	0.957	0.957	0.936	0.840
K=15	0.902	0.902	0.814	0.815	0.887	0.749	0.943	0.943	0.943	0.943	0.943	0.887	0.955	0.955	0.970	0.970	0.950	0.883
K=20	0.907	0.907	0.825	0.827	0.898	0.764	0.945	0.945	0.945	0.945	0.945	0.890	0.955	0.955	0.970	0.970	0.945	0.882
SNIPS (DS-2)																		
	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC
K=10	0.870	0.863	0.807	0.811	0.878	0.719	0.884	0.877	0.875	0.877	0.889	0.773	0.843	0.843	0.880	0.880	0.834	0.655
K=15	0.871	0.861	0.809	0.814	0.881	0.723	0.886	0.876	0.877	0.878	0.891	0.777	0.847	0.847	0.883	0.884	0.838	0.664
K=20	0.876	0.866	0.814	0.819	0.884	0.731	0.890	0.880	0.881	0.882	0.895	0.784	0.847	0.846	0.883	0.883	0.841	0.666
GOOGLE SNIPPETS (DS-3)																		
	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC
K=10	0.504	0.467	0.491	0.553	0.571	0.260	0.643	0.628	0.730	0.746	0.557	0.316	0.803	0.796	0.875	0.877	0.630	0.437
K=15	0.511	0.475	0.495	0.557	0.579	0.270	0.648	0.635	0.733	0.748	0.566	0.327	0.809	0.803	0.879	0.881	0.643	0.457
K=20	0.514	0.479	0.498	0.560	0.583	0.278	0.650	0.637	0.734	0.749	0.571	0.331	0.812	0.806	0.881	0.883	0.648	0.467
DBPEDIA (DS-4)																		
	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC	BinAcc	OpAcc	F1-S	GM1	GM2	MCC
K=10	0.924	0.924	0.938	0.938	0.915	0.840	0.965	0.965	0.980	0.980	0.923	0.828	0.984	0.984	0.992	0.992	0.938	0.755
K=15	0.926	0.923	0.942	0.942	0.914	0.841	0.970	0.969	0.983	0.983	0.936	0.849	0.986	0.986	0.992	0.993	0.927	0.782
K=20	0.920	0.918	0.934	0.935	0.912	0.832	0.962	0.961	0.978	0.978	0.923	0.817	0.983	0.983	0.991	0.991	0.928	0.739

To address query (3), we considered sets of 10, 15, and 20 in neighborhood blending to update the encoding of a particular query vector. The altered embedding for any query vector is estimated by simply performing a weighted sum of 9, 14, and 19, respectively. All the experimentation are performed in identical settings. The experimental results (Table 8) clearly state that using 20 neighbors (including itself) in the neighborhood blending approach gives better results than 10 and 15. A comparison shows that for DS-1, we have obtained 0.17% to 0.50% improved binary accuracy and open accuracy compared to sets of 10 and 15. Similarly, we have obtained 0.01% to 0.20% F1-score and GM1. In the case of DS-2, we have obtained 0.43% improved binary accuracy, 0.29% open accuracy, 0.41% to 0.52% F1-score, and 0.38% to 0.47% improved GM1. In the case of DS-3 we obtained 0.22% to 1.05% improved binary accuracy, 0.26% to 1.014% open accuracy, 0.10% to 0.34%

F1-score, and 0.08% to 0.45% improved GM1. The overall performance of the proposed work is better, with 20 of the most equivalent encoded sentences.

References

Cer, D., Yang, Y., Kong, S.-y., Hua, N., Limtiaco, N., John, R. S., ... others (2018). Universal sentence encoder. *arXiv preprint arXiv:1803.11175*.

Conneau, A., Kiela, D., Schwenk, H., Barrault, L., & Bordes, A. (2017). Supervised learning of universal sentence representations from natural language inference data. In *International conference on empirical methods in natural language processing* (pp. 670–680).

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the north american chapter of the association for computational linguistics: Human language technologies, volume 1* (pp. 4171–4186).

Iyyer, M., Manjunatha, V., Boyd-Graber, J., & Daumé III, H. (2015). Deep unordered composition rivals syntactic methods for text classification. In *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (volume 1: Long papers)* (pp. 1681–1691).

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., ... Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Reimers, N., Gurevych, I., Reimers, N., Gurevych, I., Thakur, N., Reimers, N., ... others (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. In *International conference on empirical methods in natural language processing.*

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Schroff, F., Kalenichenko, D., & Philbin, J. (2015). Facenet: A unified embedding for face recognition and clustering. In *Ieee conference on computer vision and pattern recognition* (pp. 815–823).

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems* (pp. 5998–6008).