OWTC: A Machine Learning based Framework to Identify Unseen Classes in Open-world Text classification (Supplementary File)

Appendix 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 center of all automatic document processing.

Appendix 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 modeling 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, $T_{freq}\,(t_m, d_o)$ is the frequency of term t_m.

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

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

Now IDF can be defined as ID_{freq} where

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

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 A.1 and A.2 we can define TF-IDF as

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

Appendix 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.

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

Appendix 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 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 to one another. 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 on the above network, and a keyword table is generated.

Appendix A.4. SingleRank Model for Keyword Extraction

TextRank algorithm is the most famous graph-based ranking algorithm, which is the first to use PageRank for keyword extraction. TextRank deletes all function words, and only certain parts of speech (adjective 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 A.5.

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

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 on the basis of TextRank between nodes appearing in window w at the same time. The weight is calculated by the number of times keywords occur together in the widow w at the same time. The final score of nodes V_k for SingleRank is determined by Eq. A.6.

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

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.

Appendix 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.

Appendix 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 candidate word, it uses different methods such as TF-IDF, word off-set (First and last occurrences), Altitude of terms in phrases, 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.

Appendix 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.

Appendix B. Sentence Embedding

Appendix B.1. Sentence BERT (SBERT) and different versions

SBERT extends the pre-trained Bidirectional Encoder Representations from Transformers (BERT) network [?] 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.

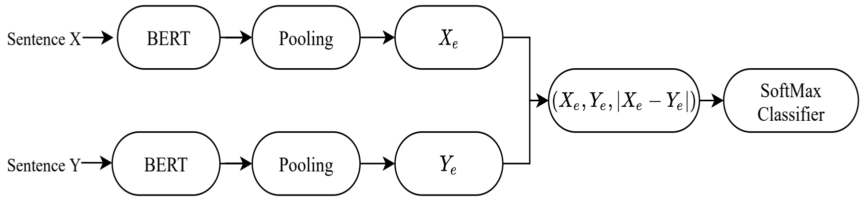

Figure B.1: SBERT framework with classification objective function where two different BERT networks have connected weights. Sentence embedding concatenate X_e and Y_e with the element-wise difference $|X_e - Y_e|$

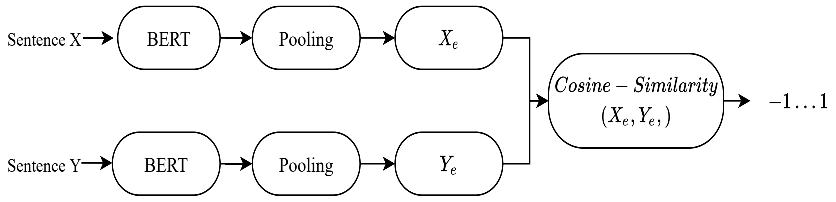

Figure B.2: SBERT framework to compute similarity scores. The objective regression function is used in this framework.

The SBERT uses siamese and triplet networks to fine-tune BERT / RoBERTa. Siamese and triplet networks [?] 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 B.1) and an objective regression function (see Figure B.2).

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 [?] instead of BERT in SBERT.

Appendix B.2. Universal Sentence Encoder

The Universal Sentence Encoder (USE) is a publicly available encoding technique 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) [**?**] and Transformer encoder [**?**]. 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 account for the sequencing and identifying of all other words in the text. The element-wise aggregation of the expressions at apiece of word location transforms the context-aware phrase representations to 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.

Appendix 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 which only actually are unseen.

$$P_{rec} = \frac{T_{p_o}}{T_{p_o} + F_{p_o}} \tag{C.1}$$

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

$$R_{ec} = \frac{T_{p_o}}{T_{p_o} + F_{n_e}} \tag{C.2}$$

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

$$ClAcc = \frac{\sum_{i=1}^{C} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}})}{\sum_{i=1}^{C} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}} + \mathrm{F_{p_o}} + \mathrm{F_{n_e}})} \tag{C.3}$$

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.

$$OpAcc = \frac{\sum_{i=1}^{C+1} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}})}{\sum_{i=1}^{C+1} (\mathrm{T_{p_o}} + \mathrm{T_{n_e}} + \mathrm{F_{p_o}} + \mathrm{F_{n_e}})} \tag{C.4}$$

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

$$BinAcc = \frac{T_{p_o} + T_{n_e}}{T_{p_o} + T_{n_e} + F_{p_o} + F_{n_e}} \tag{C.5}$$

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}

$$F1\text{-}score = 2 \times \frac{P_{rec} \times R_{ec}}{P_{rec} + R_{ec}} \tag{C.6}$$

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 predicts class values, and 1 indicates the absolute agreement between actual class values and predicted class values. It is calculated as:

$$MCC = \frac{(T_{p_o} \times T_{n_e}) - (F_{p_o} \times F_{n_e})}{\sqrt{(T_{p_o} + F_{p_o})(T_{p_o} + f_{n_e})(T_{n_e} + F_{p_o})(T_{n_e} + F_{n_e})}} \tag{C.7}$$

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}

$$GM1 = \sqrt{P_{rec} \times R_{ec}} \tag{C.8}$$

$$GM2 = \sqrt{R_{ec} \times S_{peci}} \tag{C.9}$$

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

$$ARS = \frac{(RI - ExpectedRI)}{(max(RI) - ExpectedRI)} \tag{C.10}$$

   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:

$$NMI(g_i, g''_j) = \sum_{i=1}^{|g_i|} \sum_{j=1}^{|g''_j|} \frac{|g_i \cap g''_j|}{N} \log \frac{N|g_i \cap g''_j|}{|g_i||g''_j|} \tag{C.11}$$

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

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

$$FMS = \frac{T_{p_o}}{\sqrt{(T_{p_o} + F_{p_o}) * (T_{p_o} + F_{n_e})}} \tag{C.12}$$