Combination of word embeddings for long text classification

Apoorv Awasthi, Bala Akhil Rajdeep Battula
Indiana University-Purdue University, Indianapolis, IN 46202, USA
aawasth@iu.edu, bbattula@iu.edu

1. Introduction

Word embedding is a technique to represent text as a vector. The vectors created by word embedding techniques are multidimensional arrays that contain information about the text in every dimension. Each dimension in an array contains a numerical value that conveys information about a particular feature of the text.

There are mainly two types of word embeddings. The first is the traditional word embedding. These word embeddings represent each word as a vector. These vectors are chosen such that there is semantic and syntactic relationships between the words and the vectors can be used to find semantically or syntactically similar word. An example of traditional word embeddings is the Word2Vec embedding technique.

The other type of word embedding is contextual word embedding. These embeddings can provide different vectors to the same word based on the context of the sentence in which the word appears. Moreover, these word embedding techniques can generate a single vector for the whole sentence. Some examples of contextual word embedding include BERT and ALBERT word embedding techniques.

In this paper, we are using BERT along with other word embedding techniques for long texts and then training a classification model and evaluating its performance. We also provide some preliminary results in this paper showcasing how combining vectors formed by various word embeddings can result in better accuracy for the classification model.

2. Problem description

Contextual word embedding techniques are used to create a single vector for the whole sentence. BERT-base converts the sentence into a vector of 768 dimensions in which each of the dimensions has a numerical value telling about a particular feature. If the sentence length increase, the sentence will have even more information which will be very difficult to be conveyed by a single vector. Also, BERT is incapable of processing long text due to time consumption as well as quadratically increasing memory[1].

BERT can only accept less than 512 tokens[1]. There are a few techniques used to treat long text. We can use the first 510 tokens, the last 510 tokens, or the first 128 tokens and the last 382 tokens together.[2] But using these truncation methods will reduce valuable information that we obtain from the text and thus can result in an incorrect vector.

Another way to deal with the problem is to split the text into 512 token sections and use BERT for all the separate sections. But again, this method is time-consuming and memory intensive.

In this research paper, we introduce a simple yet effective way to produce word embeddings for long text using BERT in combination with other word embedding techniques. Doing it this way won't reduce any valuable information and is less time-consuming and less memory intensive. This is the novel idea of this project

3. Dataset used

The dataset is taken from Kaggle. https://www.kaggle.com/datasets/venky73/spam-mails-dataset?select=spam_ham_dataset.csv

This dataset contains information about spam and non-spam emails. There are three columns in this dataset:

- Label: This column tells us whether the mail is spam or not
- Text: This column contains the actual text of the email.
- label_num: This column is the same as the label column but it contains boolean values (0 and 1) to indicate whether the email is spam or not. 0 means not spam whereas 1 means spam.

We use this dataset to do text classification and predict whether the email is spam or not.

4. Methodology

4.1 Data cleaning

The first step that we did was the data cleaning step. Our dataset had the text column which had '\r' and '\n' characters. We removed them in this step.

text
Subject: enron methanol ; meter # : 988291\r\n...
Subject: hpl nom for january 9 , 2001\r\n(see...
Subject: neon retreat\r\nho ho ho , we ' re ar...
Subject: photoshop , windows , office . cheap ...
Subject: re : indian springs\r\nthis deal is t...

Fig 1: Text previously

clean_text
Subject: enron methanol ; meter # : 988291. th...
Subject: hpl nom for january 9 , 2001. (see a...
Subject: neon retreat. ho ho ho , we ' re arou...
Subject: photoshop , windows , office . cheap ...
Subject: re : indian springs. this deal is to ...

Fig 2: Text after cleaning

4.2 Subsampling of data frame into 500 rows

The second step was to reduce our data frame to 500 rows with each label having 250 rows. We did this step due to computing limitations and just to showcase some preliminary results.

4.3 Calculating the length of text for each row

The next step was to see the sentence length of the text column. We obtained the text length for each row and used the describe function to get the summary.

count	500.000000
mean	1201.890000
std	1730.086265
min	11.000000
25%	273.000000
50%	612.000000
75%	1326.750000
max	14716.000000

Fig 3: Summary of text lengths

We see in the summary that the average length of the text column is 1201 words. This value is way above the maximum tokens which BERT can accept (512 tokens)

4.4 Obtaining the word embedding vector using BERT-base

We use the BERT-base model to obtain word embeddings or vectors having 768 dimensions for each text value. Then we convert these vectors into a numpy array.

4.5 Obtaining the word embedding vector using ALBERT

We use the ALBERT model to obtain word embeddings or vectors having 768 dimensions for each text value. Then we convert these vectors into a numpy array as well.

4.6 Obtaining the word embedding vector using Word2Vec

Word2vec word embeddings are meant for a single word. It will provide a vector for each word in the sentence. To obtain word embedding for the whole sentence, we first preprocess the text and convert it into a list of words in that sentence which will be fed as an input to the word2vec embedding. Word2vec will provide an array containing vectors from each word in the sentence. So for each sentence, we will have many vectors inside an array. We then average the respective elements of all the vectors in a single sentence to obtain a single vector for each sentence. We kept the vector size at 100.

4.7 Combining word embedding vectors.

In the next step, we combine all the word embedding vectors in all possible combinations. All the vectors obtained from word embedding techniques are numpy arrays and we combine these arrays by concatenation. We obtain four more vectors namely: BERT+w2v, ALBERT+w2v, BERT+ALBERT, and BERT_ALBERT+w2v.

w2v_output	BERT_output_array	ALBERT_output	ALBERT_output_array	BERT+w2v	ALBERT+w2v	ALBERT+BERT	ALL
[-0.7773241, -0.028775526, 0.23354693, 0.34879...	[-0.66220784, -0.21042265, -0.69894105, 0.4461...	(tf.Tensor(0.53995717, shape=(), dtype=float32...	[0.53995717, -0.56981885, -0.98754895, -0.3067...	[-0.662207841873169, -0.21042264997959137, -0....	[0.5399571657180786, -0.5698188543319702, -0.9...	[0.5399571657180786, -0.5698188543319702, -0.9...	[0.5399571657180786, -0.5698188543319702, -0.9...
[-0.81515855, -0.042400777, 0.2503332, 0.38154...	[-0.62916726, -0.43002403, -0.748989, 0.363675...	(tf.Tensor(0.6074936, shape=(), dtype=float32)...	[0.6074936, -0.6763085, -0.96309423, 0.0338184...	[-0.6291672587394714, -0.43002402782440186, -0...	[0.6074935793876648, -0.6763085126876831, -0.9...	[0.6074935793876648, -0.6763085126876831, -0.9...	[0.6074935793876648, -0.6763085126876831, -0.9...
[-0.82275397, -0.026751796, 0.24303724, 0.3726...	[-0.60679173, -0.3678604, -0.77422667, 0.46082...	(tf.Tensor(0.7890521, shape=(), dtype=float32)...	[0.7890521, -0.8926711, -0.98849905, -0.099861...	[-0.6067917346954346, -0.3678604066371918, -0....	[0.7890521287918091, -0.8926711082458496, -0.9...	[0.7890521287918091, -0.8926711082458496, -0.9...	[0.7890521287918091, -0.8926711082458496, -0.9...
[-0.94489324, -0.028019296, 0.27957302, 0.4287...	[-0.5414106, -0.54300076, -0.9778723, 0.620954...	(tf.Tensor(0.4502093, shape=(), dtype=float32)...	[0.4502093, -0.5150742, -0.997151, -0.61924064...	[-0.5414106249809265, -0.5430007576942444, -0....	[0.45020928978919983, -0.5150741934776306, -0....	[0.45020928978919983, -0.5150741934776306, -0....	[0.45020928978919983, -0.5150741934776306, -0....
[-0.652256, -0.027082026, 0.1878234, 0.2854014...	[-0.7903552, -0.58141476, -0.9750504, 0.643725...	(tf.Tensor(0.629446, shape=(), dtype=float32),...	[0.629446, -0.73356396, -0.9876126, 0.06124072...	[-0.7903552055358887, -0.5814147591590881, -0....	[0.6294460296630859, -0.7335639595985413, -0.9...	[0.6294460296630859, -0.7335639595985413, -0.9...	[0.6294460296630859, -0.7335639595985413, -0.9...

Fig 4: Image showcasing all the vectors obtained

4.7 Classification models.

Next, we use all these vectors separately as input to the random forest classification model and try to predict the label. We had seven vectors namely: BERT, ALBERT, Word2vec, BERT+word2vec, ALBERT+word2vec, BERT+ALBERT, and BERT+ALBERT+word2vec. Using all these vectors as input separately, we train seven random forest models. Results are showcased in the preliminary results section.

5. Preliminary results

We used BERT and various combinations of word embeddings to create seven vectors for each long sentence. Using those vectors, we tried to achieve high accuracies on the long text classification problem. The results here include only the results of this particular task.

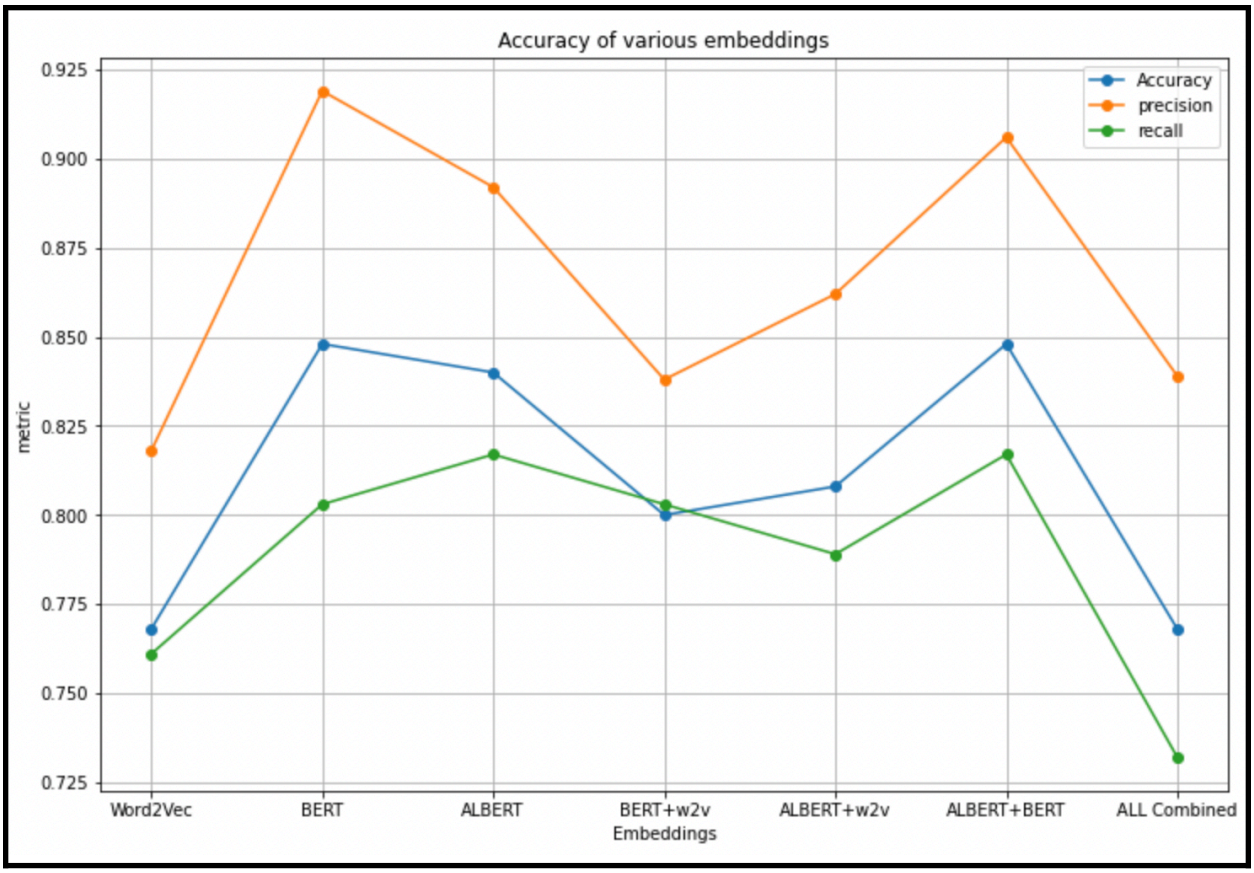

Fig 5: Plot of accuracy, recall, and precision for all the models using various word embedding combinations

As you can see from the above plot, if we train the classification model using vectors produced from BERT, we get the highest precision but low recall. But if we use vectors created using the ALBERT model, we get a high recall. W2v model gives low values of recall, accuracies, and precision individually as well as in combination with other vectors. A combination of BERT and ALBERT vectors showcased interesting results. Models using BERT vectors had high precision whereas models using ALBERT vectors had high recall. When combining vectors from both the word embeddings, we get a vector having 1536 dimensions. This new vector contains properties from both BERT and ALBERT vectors. BERT vector contributes towards precision whereas ALBERT contributes towards recall. The accuracy obtained when combining BERT and ALBERT vectors was 0.85 which was the highest as compared to other vectors. The heat map and classification reports are shown below:

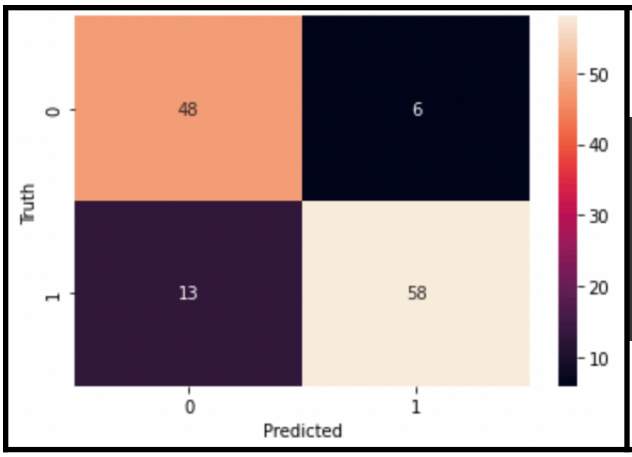

Fig 6: Heatmap of the best classification model

	precision	recall	f1-score	support
0	0.79	0.89	0.83	54
1	0.91	0.82	0.86	71
accuracy			0.85	125
macro avg	0.85	0.85	0.85	125
weighted avg	0.85	0.85	0.85	125

Fig 7: Classification report of the best classification model

These results are only for a particular task of long text classification. We can use the combination of BERT along with various word embeddings to achieve high performance on other NLP tasks as well.

6. Evaluation plan

We plan to extend this method of combining various word embeddings on other NLP tasks as well to test whether it can capture the properties of the long text. But, at first, we plan to do the same classification problem on the full dataset. For evaluating the success of our word embeddings, we will use the final evaluation metrics of the classification model like - accuracy, recall, and precision and also a heat map to visualize how the model performed as we showcased above. If the model shows high accuracies, that means that the word embeddings created can capture most of the properties of the sentence, and thus the vector can determine the label.

For other NLP tasks, we plan on creating the vectors and then evaluating the embedding performance based on the NLP task evaluation metrics which vary with different tasks. These evaluation metrics may be bilingual evaluation understudy (BLEU), MAPE (Mean Absolute percentage error), perplexity, etc.

7. Expected outcomes

As showcased in the preliminary results, we achieved high accuracy on the long text classification task by combining two vectors formed by different word embedding techniques. If we extend this to other NLP tasks as well, we may get good performance. We plan to try other combinations as well. For example: for long scientific texts we can try sciBERT and also a combination of sciBERT with other word embedding techniques to capture more details of the long sentence. By combining vectors, we are striving to capture more information about long texts by adding more dimensions. Having more features can prove beneficial if we are dealing with long texts.

8. References

1) *CogLTX: Applying Bert to long texts - neurips*. (n.d.). Retrieved July 29, 2022, from https://proceedings.neurips.cc/paper/2020/file/96671501524948bc3937b4b30d0e57b9-Paper.pdf

2) *How to fine-tune bert for text classification? - arxiv*. (n.d.). Retrieved July 29, 2022, from https://arxiv.org/pdf/1905.05583.pdf