# NLP Homework – 3

* Supreeth B S

Word2Vec Task:

* Upon reading about word2vec, learnt that there are two approaches, one being the continuous bag of word (CBoW) and the skip-gram approach.
* It took a lot of time understanding the usage of word2vec and it’s working. As I am not well versed with these methods, It took a lot of time studying and trying different approaches.

Initially I took a wrong approach where I thought we were required to get the top k most similar words from both files, instead of the top k most occurring words. I have left the commented code just for logging the approach.

For this task, I first took the data and split the data into “OFF” and “NOT” type of data. Later upon passing the data to the compare\_text\_word2vec function. Here the, I first found the top k occurrences from each subset using counter. Before anything happened with the data, data cleaning is done by clean\_tweet function, which removed all the punctuations, stop words, lowering the case of all the words. Upon getting the two list of words.

For training the word2vec I take the sentences of the tweets and use the words (labeled:data in the code) is passed to the word2vec. I use vector\_size as 10.

I encountered lots of issue trying to get the data in the right format for training the word2vec. The approach I had taken was not yielding the right results in terms of training the model. The final approach is finally submitted to you.

For part 1: I used ‘glove-twitter-25’ pretrained word2vec model present in the genism. Also, I have trained the model with the data present in the training data which will later be used in the second part of the assignment. This model is trained with window sizeof 50 and running for epochs of 200. It generated better results in MLP part of the assignment which we will come to at the later part of the report.

Using the word2vec model, I use the model.wv.similarity for finding the similarity score between the two words from the list of OFF\_data and NOT\_data. I got all the scores for the top\_K words. Got the average between all the words, indicating the similarities for the documents based on the top\_k most occurring words.

Similarly, I used the cosine similarity between the documents using the topK words. The cosine\_similarity function is imported from the sklearn.metric.pairwise. It takes two vectors’ embeddings of the words that needs to be used for finding the similarities. Found out later that both the cosine\_similarity from sklearn and word2vec.similarity gives the cosine similarity so have presented only one of it.

I explored different approaches to find similarities such as softCosineSimilarity, annoyindexer.{I have left the part of the code in the word2vec submission so that it I can preserve my attempts of different approaches.

Result discussion:

* + Upon seeing the list of most occurring words in both subset after removing the stop words: I found out that there is a lot of words repeating in both the subset. Indicating that the words present in both offensive and non-offensive data.
  + Some words such as liberals, gun, antifa, people…. Were present in both of the subsets. I expected more distinct words to be present in each subset without having such high similarity.
  + For k=5, the similarity score was found to be approximately 44% between the files.
  + For k=10, the similarity score was found to be approximately 48% between the files.
  + For k=20, the similarity score was found to be approximately 63% between the files.
  + I found some of the similar words that we get from pre-trained model is better suggested when used with word2vec(trained on the training data) rather than the pre-trained word2vec(Twitter-25).

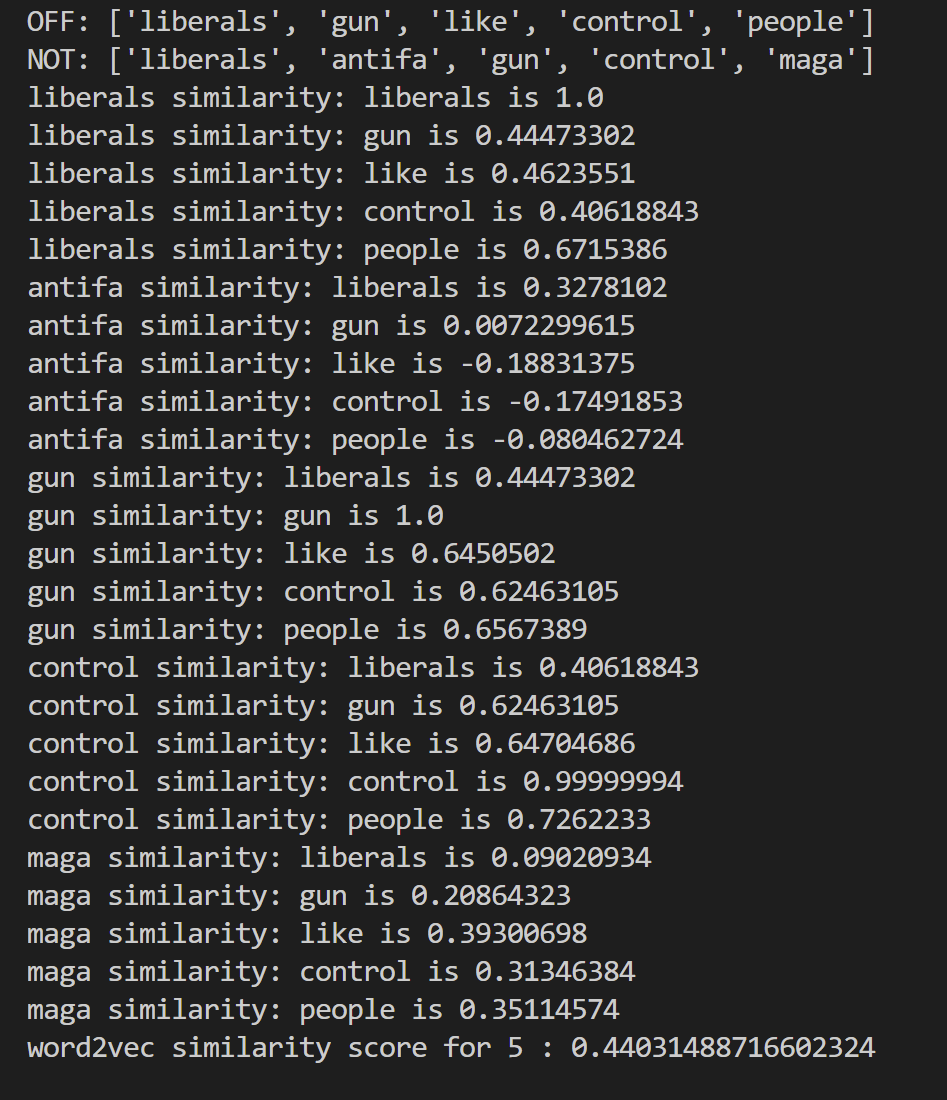
NOTE: The screenshots are few important parts of the results, for full results (If needed, please run the program).

NOTE2: I have left some part of the unused code as that was one of my attempts for this assignment that I wanted to keep in the submission.

NOTE BEFOR RUNNING: Please change the location of the training file for the training dataset.

The similarity score first prints the whole list(matrix of k x k) of the similarity scores. Finally, the similarity score for the files are shown.

Similarity score for k=5 (Full output has been given) Here we see the score of each word’s similarity score has been printed, and finally the cosine similarity score is shown.(Highlighted in the image)





Similar word list for k=5

Text

Description automatically generated

Text

Description automatically generated with low confidence

Below is the list of topk words where k=10. Similarity score for k=10(only a snippet of the score has been put here. For full list of similarity score if needed, please run the code.)

Text

Description automatically generated

Text

Description automatically generated



Text

Description automatically generated

Below is the list of topk words, where k=20. Similarity score here is 62%. This is just snippets of important part

along with list of few similar word list.

Text

Description automatically generated

Text

Description automatically generated



Text

Description automatically generated

Text

Description automatically generated with medium confidence

MLP ASSIGNMENT PART:

NOTE: To run the file please specify the training and testing files in line 102,103. Also the labels for test files in line 86.

* Cleaning the data is done using the same clean function that is removing all the punctuations and stop words, lowering the cases.
* In training of the MLP I passed the number of layers required in that iteration as it is required for us to do it for 1,2,3 layers.
* I have saved the model from word2vec part of assignment as well as I have done it again just toget it in the assignment again.
* I have used create\_vector and MEV function for getting the embeddings of word for input tweets from test and train data.
* create\_vectors takes all the tweets, it further tokenizes to sentences and further down to word level tokens.
* From the vectors created in the create\_vectors functions, the MEV function creates embeddings by taking the mean of all the vectors present in the vocab.

NOTE: [EXTRA CREDIT] update\_embeddings: This function trains the model using the data from the training dataset, and the following parameters were used to create.

w2v1\_all = Word2Vec(sentences = data,vector\_size = 10,window = 50,epochs = 200)

I have used the self-trained model with updated embeddings here as it was giving better results and also some keys didn’t have embeddings in the pretrained model, so it worked in this. For updated embeddings, I have used the data from the training data for the training of word2vec training which gave better word embeddings for the MLP model’s better accuracy.

OUTPUT:

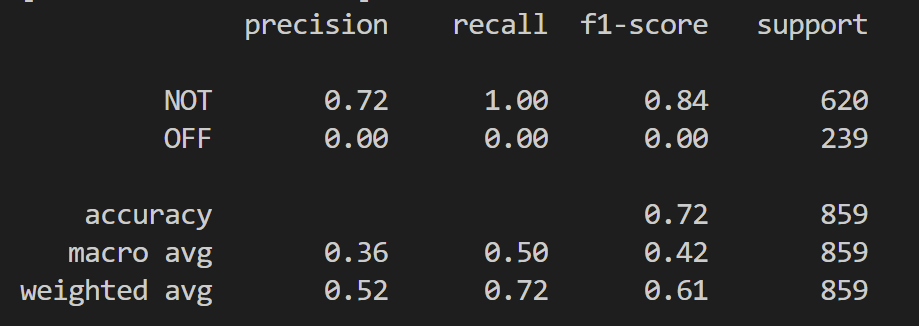
Results for layer 1: I got 72% precision for NOT type of data, but surprisingly it gave 0 for OFF type of data. Over all precision was given as 72, which might not be completely correct as there might be overfitting of NOT type of data in the model.

Result for layer 2: I got overall accuracy of 74% which is better than layer=1 result as there is better precision for OFF and NOT type of data.

Result for layer 3: I got an overall accuracy of 75% which is almost similar to layer=2 but the precision of OFF type has increased from 66% to 72%.

TEST\_MLP creates 3 csv files with labels(MLP1\_output,MLP2\_output,MLP3\_output) which is the output csv files for different MLP models with k layers.

LAYERS = 1



LAYERS = 2

A screenshot of a computer

Description automatically generated with low confidence

LAYERS = 3

