**Extra Credit 2 Report**

Name: Prajwala Mugajji Shambulingappa

**Aim:** Training my own word2vec embeddings and combine them with simple neural networks for classification.

**Observation Criteria and Analysis:**

1. Packages Used: nltk, re, pandas, NumPy, pickle, nltk - 'stopwords', nltk -'punkt', pickle, sklearn – ‘f1\_score’ and ‘accuracy\_score’, sklearn – ‘MLPClassifier’ and genism – ‘downloader’.
2. Dataset: Dataset used for is provided by Kaggle which has a large number of Wikipedia comments that have been labelled by toxic behavior by human reader.
3. Pretrained model used: word2vec-google-news-300.
4. Preprocessing: As part of preprocessing, I have created tokens and normalized these tokens by lowercasing, removing whitespaces, splitting, and removing empty strings. After which, I removed the stopwords using nltk package.
5. Generating Embeddings: As an extension of the preprocessing, I moved on to generate embeddings for the training data using the word2vec model. While performing this step, I faced issues such as some words didn’t have any embeddings in the pretrained model for which I appended a vector of zeros with length max\_length that was set to 300. In addition to that, I constantly faced RAM issues as loading all the data into the processor at once and training the model kept crashing my system, to tackle that I made all the embedding to be the same length, that is the embeddings were padded with zeros if they are shorter than max\_length or truncated if longer. Then the resulting embeddings were returned as a NumPy array, and the average of the vectors were taken as the final embedding for the input text. This approach worked well for me and gave me comparatively better accuracies while training and testing, Thus, I choose to continue with the same approach.
6. Update Embeddings: This method fine-tunes a pretrained word2vec embedding on the training text and then saves the fine-tuned embeddings. While updating the embeddings, initially fearing the RAM issue I tried to divide the training data into small chunks/batches but this approach took way to long to just update embeddings, to reduce that I tried to used Fast Text, an NLP package that allows us to train supervised and unsupervised representations of words as well as sentences while increasing the batch size, although this was a good approach due to my RAM restrictions I couldn’t really leverage Fast Text. At the end, I choose to use the ‘build\_vocab’ method to update the vocabulary of the new model, that I will be using to fine-tune, to include any new words from the pre-trained model. Then used ‘intersect\_word2vec\_format’ method to update the word vectors of the new model to include any shared words with the pre-trained model. Finally, used the ‘lockf’ parameter to specify the fraction of the pre-trained model’s weights that were kept fixed during the entire training.
7. Training the model: This method uses Multi-Layer Perceptron (MLP) as the training model using pre-trained Word2Vec embeddings as well as my fine-tuned embeddings. The code then evaluates it and reports the accuracy and F1 score.
8. Testing the model: This method tests a trained MLP model on a test file.

**Summary:**

1. Method that trains a multi-layer perceptron model on given training data leveraging a pre-trained Word2Vec embeddings as well as my own word2vec embeddings and returns that model was implemented successfully.
2. Tabulating the training accuracy for different layers of the MLP model and different learning rates.
   1. For pretrained embeddings:

|  |  |
| --- | --- |
|  | Train accuracy |
| num\_layer = 1 | 0.9542 |
| num\_layer = 2 | 0.9548 |
| num\_layer = 3 | 0.9568 |

* 1. For my embeddings:

|  |  |
| --- | --- |
|  | Train accuracy |
| num\_layer = 1 | 0.9562 |
| num\_layer = 2 | 0.9582 |
| num\_layer = 3 | 0.9559 |

1. Tabulating the test f1 scores for MLP models with different layers.
   1. For pretrained embeddings

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Learning rates | Test accuracy | Test-F1 score | Model F1 score | Model accuracy |
| num\_layer =1 | 0.005 | 0.9226 | 0.6359 | 0.6359 | 0.9226 |
| num\_layer = 2 | 0.005 | 0.9152 | 0.6369 | 0.6368 | 0.9151 |
| num\_layer = 3 | 0.005 | 0.9275 | 0.6456 | 0.6455 | 0.9274 |

* 1. For my embeddings:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Learning rates | Test accuracy | Test-F1 score | Model F1 score | Model accuracy |
| num\_layer =1 | 0.005 | 0.9251 | 0.6411 | 0.6411 | 0.9250 |
| num\_layer = 2 | 0.005 | 0.9262 | 0.6450 | 0.6450 | 0.9262 |
| num\_layer = 3 | 0.005 | 0.9340 | 0.6387 | 0.6386 | 0.9339 |

NOTE: The google drive link to the fine-tune embeddings  
https://drive.google.com/file/d/1yohIxcjkmtAMiN0pirOexrJQlMjfVvbx/view?usp=share\_link