**Assignment Report**

Name: Prajwala Mugajji Shambulingappa

**Aim:** Incorporating word2vec embeddings with simple neural networks for classification.

**Observation Criteria and Analysis:**

1. Packages Used: nltk, re, pandas, NumPy, csv, 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 followed multiple methods. Firstly, instead of creating a list of word vectors I chose to implement a 3-D numpy array, which did help but my session kept crashing. After which I tried to pass the data in batches, which again didn’t help me much, however that only increased the training time significantly as I had to give a very small batch size. Moving on, I tried to lower the word-embedding dimensions, this solved the problem, but the model performance was significantly poor. Finally, 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 good accuracies while training and testing, Thus, I choose to continue with the same approach.
6. Training the model: This method uses Multi-Layer Perceptron (MLP) as the training model using pre-trained Word2Vec embeddings. The code then evaluates it and reports the accuracy and F1 score.
7. 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 and returns that model was implemented successfully.
2. Tabulating the training accuracy for different layers of the MLP model and different learning rates.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Learning rate: 0.0005 | Learning rate: 0.005 | Learning rate: 0.05 | Learning rate: 0.5 |
| num\_layer = 1 | 0.9488 | 0.9533 | 0. 9500 | 0.9375 |
| num\_layer = 2 | 0.9538 | 0.9562 | 0.9527 | 0.9042 |
| num\_layer = 3 | 0.9545 | 0.9575 | 0.9534 | 0.9042 |

1. Observation with model accuracy while training:

The model’s performance is slightly better when we increase the number of layers in the MLP model with a learning rate of 0.005.

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Learning rates | Test accuracy | Micro-F1 score | Macro-F1 score | Weighted-F1 score |
| num\_layer =1 | 0.0005 | 0.9244 | 0.9244 | 0.7815 | 0.9245 |
| 0.005 | 0.9294 | 0.9294 | 0.7990 | 0.9301 |
| 0.05 | 0.9156 | 0.9156 | 0.7905 | 0.9216 |
| 0.5 | 0.9281 | 0.9281 | 0.7269 | 0.9167 |
| num\_layer = 2 | 0.0005 | 0.9237 | 0.9237 | 0.7969 | 0.9268 |
| 0.005 | 0.9173 | 0.9173 | 0.7932 | 0.9229 |
| 0.05 | 0.9290 | 0.9290 | 0.7939 | 0.9290 |
| 0.5 | 0.9048 | 0.9048 | 0.4750 | 0.8596 |
| num\_layer = 3 | 0.0005 | 0.9238 | 0.9238 | 0.7982 | 0.9271 |
| 0.005 | 0.9257 | 0.9257 | 0.8038 | 0.9290 |
| 0.05 | 0.9269 | 0.9269 | 0.7974 | 0.9285 |
| 0.5 | 0.9048 | 0.9048 | 0.4750 | 0.8596 |

1. Observation with model accuracy while testing:

The model’s performance is slightly better when we increase the number of layers in the MLP model with a learning rate of 0.005. However, a learning rate of 0.5 gives the worst performance.