Multi-class Text Classification of Public Medical Notes with LSTM

WHEELER LI

Within this mini-project, a natural language processing machine-learning pipeline is built to achieve the classification of the abstracts of Public Medicine into its corresponding Medical Subject Headings. The machine learning network based on the Long short-term memory (LSTM) model is implemented for the classification. At the same time, pre-trained 'GloVe' and 'FastText' word embedding models are used for word vectorisation. 74% and 71% of test accuracy was reached with the FastText and GloVe model respectively. The potential improvements are also discussed in the report.

Additional Key Words and Phrases: NLP, LSTM network, PubMed, Classification

ACM Reference Format:
Wheeler Li. 2023. Multi-class Text Classification of Public Medical Notes with LSTM . *J. ACM* 37, 4, Article 111 (August 2023), 7 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

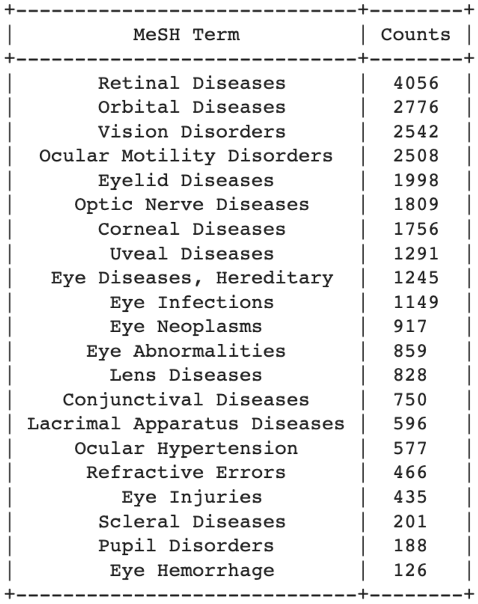
In natural language processing (NLP), Multi-label classification (MLC) is a significant task that can be used in various practical scenarios[1], including tag recommendation, information retrieval and the pipeline in this project is able to be adapted to clinical letter classification. The abstract texts are obtained from PubMed, a publicly available database of biomedical literature. The aim of this project is to classify abstracts from ophthalmology papers into their Medical Subject Headings. The implementation of neural networks in the field of NLP has brought significant success over recent years. Different types of approaches are applied including CNN, KNN and RNNs. The LSTM network is used in this task since this network is able to learn long-term dependencies of words in a context while networks such as CNNs were designed to capture local patterns within a fixed-size window. Hence, the association between labels or differences in textual content are often neglected[1]. The whole pipeline is separated into the following stages: data exploration, data cleaning and tokenization, embedding layer set-up and model training.

2 METHOD

2.1 Data Overview

The data from the CSV document is imported into the Pandas dataframe in Python. 27073 rows of samples and 21 different classes are found in total. The occurrences of classes are counted and a significant imbalance needs to be noted. From Fig.1, Retinal Diseases is the class that appears the most often, with 4056 counts, while Eye Hemorrhage and Pupil Disorders only appear less than 200 times. This significant difference in occurrences will likely cause an imbalance of test accuracy in the model output (more accurate for the classes that appear more often). The number of words in each text is also reviewed with a density plot.

Author's address: Wheeler Li, zcapicx@ucl.ac.uk.



```
+------------------------------+--------+
|           MeSH Term          | Counts |
+------------------------------+--------+
|       Retinal Diseases       |  4056  |
|       Orbital Diseases       |  2776  |
|       Vision Disorders       |  2542  |
|  Ocular Motility Disorders   |  2508  |
|       Eyelid Diseases        |  1998  |
|     Optic Nerve Diseases     |  1809  |
|       Corneal Diseases       |  1756  |
|        Uveal Diseases        |  1291  |
|   Eye Diseases, Hereditary   |  1245  |
|        Eye Infections        |  1149  |
|        Eye Neoplasms         |  917   |
|      Eye Abnormalities       |  859   |
|        Lens Diseases         |  828   |
|     Conjunctival Diseases    |  750   |
| Lacrimal Apparatus Diseases  |  596   |
|      Ocular Hypertension     |  577   |
|      Refractive Errors       |  466   |
|         Eye Injuries         |  435   |
|       Scleral Diseases       |  201   |
|       Pupil Disorders        |  188   |
|        Eye Hemorrhage        |  126   |
+------------------------------+--------+
```

Fig. 1. The Occurrences of Each MeSH Class

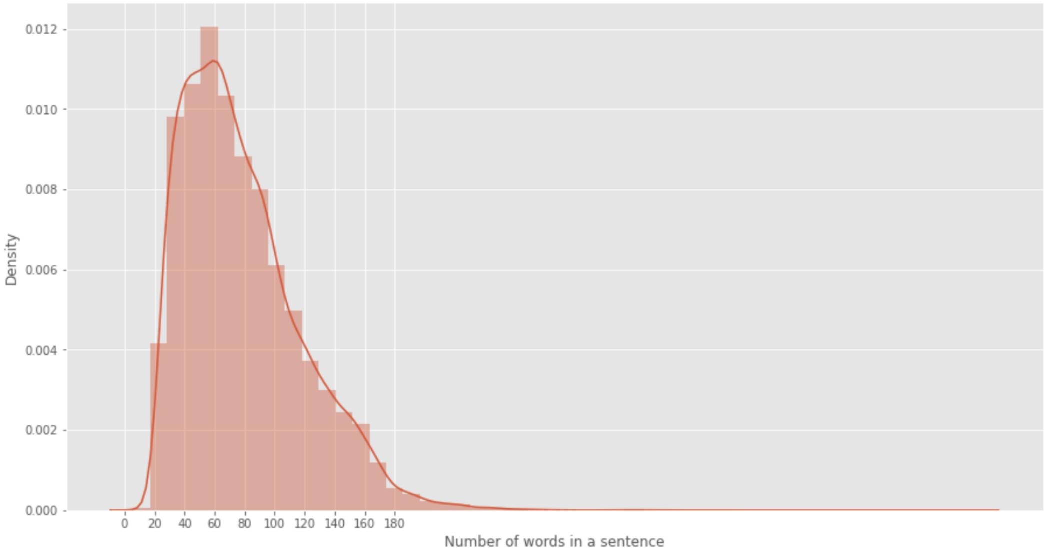


Fig. 2. .The Density Distribution of Number of Words in Sentences (Tokenised)

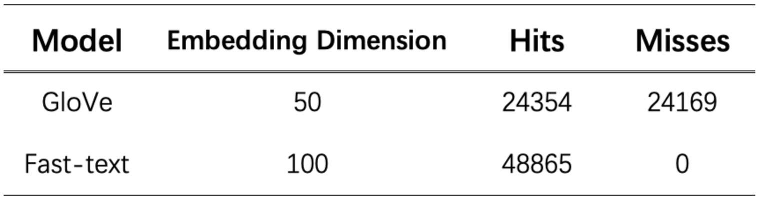
According to Fig.2, most of the sentences consist of 40 to 80 words, this number provides an indicator of the setting of the maximum length of the text sequence.

2.2 Data Pre-processing

To deal with complete sentences in abstract texts, cleaning and tokenisation are taken to break them down into smaller units such as single words and individual characters. The regular expression and ’stopword’ from nltk are imported. Characters such as single alphabets, numbers, and plus or minus signs are excluded from the sentences and special characters like brackets are replaced with spaces. Then, each word left in a sentence would be separated by a space. The distribution of the tokens is investigated using the ’FreqDist’ function in the nltk module and I found that the frequency that each token appears is relatively even (2 or 3 times). Therefore, this distribution roughly reflects successful tokenisation and ensures that the text is not dominated by a small number of highly frequent words. The MeSH labels are one-hot encoded into 1-d arrays with sizes of 21 using the ’get dummies’ function provided by Pandas.

2.3 Embedding Models

Several pre-trained word embedding models are imported from gensim. The GloVe and FastText embeddings are tested in this task. Firstly, the TextVectorization in Tensorflow is used to index the vocabulary of the raw texts and allows the text data to be converted into feature vectors.[2] All of the tokens are considered (48865 words in total) here and the output sequence length is set to 64. This sequence length reflects the number of tokens in each sentence. Then, the embedding matrices are created. For each token in a sentence, a matrix only contains 0 with a size equal to the embedding dimension is created, hence, for any sentences with shorter length, the remaining spaces would be considered as a zero matrix. At the same time, for sentences with longer lengths, only the first 64 tokens are included and inputted into the embedding layer. Then, the embedding matrices are created from the model and the token indices. Each token is searched within the model and the corresponding word vector would be assigned in the embedding matrix and be marked as ’hit’, any token that is not found in the model would not be assigned any value and marked as ’miss’, so it would remain as a zero-matrix(created by np.zeros).



Model	Embedding Dimension	Hits	Misses
GloVe	50	24354	24169
Fast-text	100	48865	0

Table 1. Number of Word to Vector Conversions

From table 1, we can find that the Fast-text model contains all of the tokens while nearly half of the tokens are missed in the GloVe model (including duplicates). This also suggests a better performance might be observed with the Fast-text embedding.

2.4 LSTM Networks and Evaluation

The construction of the neural network contains an embedding layer (which loads the pre-trained embedding matrix), and an LSTM layer to process the sequence of vectors and selectively retain information. Then, a fully-connected dense layer is used for dimension reduction and produces the final output. Since there are 21 classes in total, the output shape of the dense layer is also set as 21. The parameter being adjusted is the training batch size and the number of neurons in the LSTM layer. And the initial batch size is set to 256 while 64 neurons are put in the LSTM layer. The performance of the neural network is evaluated by the accuracy and categorical crossentropy loss function for this multi-class classification task. In categorical cross-entropy loss, the softmax output of the network would produce the predicted probability distribution, while the actual probability distribution as reference is represented by one-hot encoded labels defined previously. The value of loss function is calculated as:

$$Loss = -\sum_{c=1}^{M} y_{o,c} \, . \ln(p_{o,c}) \tag{1}$$

Where M is the number of classes (21 in this task), y is a binary indicator (0 or 1) if class label(c) is the correct classification for observation(o) and p is the predicted probability. The separate loss is calculated for each label in observation and the results are summed up. [3] The accuracy and loss values are plotted for the training and validation data sets. At the same time, the Drop out is set to 0.7 in the LSTM layer, so that the network would randomly dropping out neurons to prevent overfit.

3 EXPERIMENT AND RESULTS

In this section, two different embedding models are tested and the 'Fast text' model achieves a slightly higher accuracy. The size of batch size is tuned and the test result from 3 different batch sizes are displayed. The number of epochs is set to 25 and the validation accuracy is monitored in each epoch. Since over-fitting is observed in each network, the number of neurons in the LSTM layer is not adjusted (stay at 64), since more neurons are likely to increase the effect of over-fitting. At this stage, the test data size is set to 0.2 and another 20% of training data is set for validation monitoring.

3.1 GloVe Model

For the Glove Model, the batch size and the neuron number of LSTM are set to 64. The Glove matrix is imported into the Embedding layer, which is set as the first layer of the neural network.

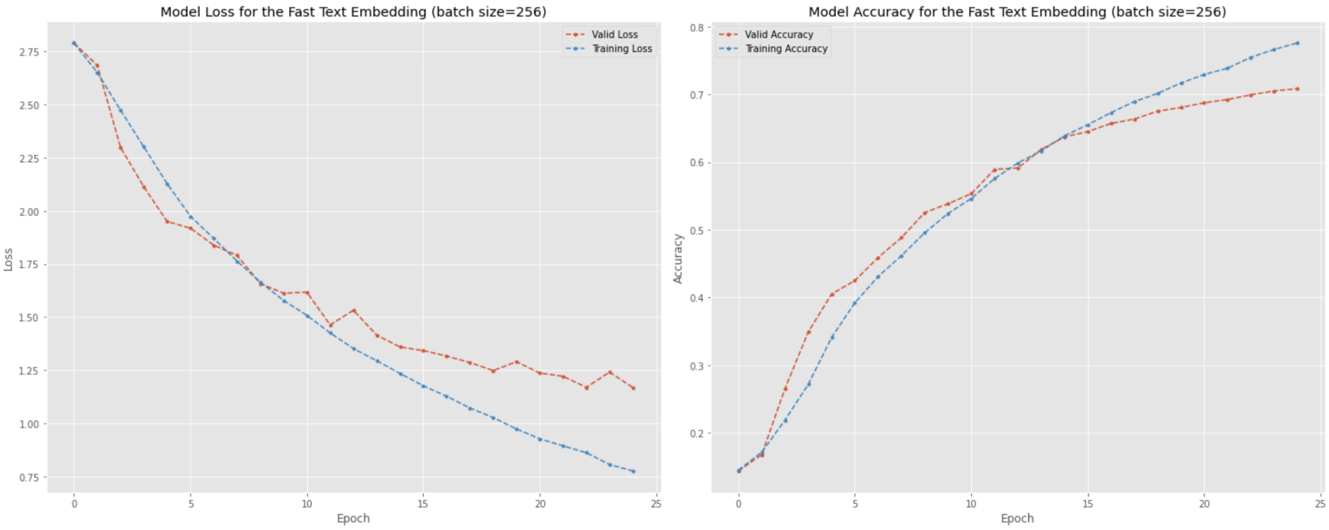
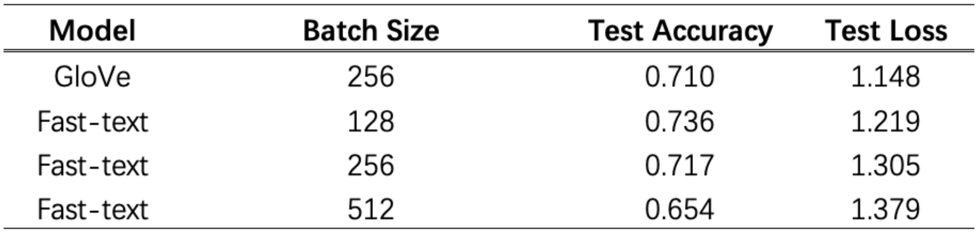


Fig. 3. Performance of the LSTM Network with GloVe Embedding

From Fig.3, the overfitting effect becomes significant after the 10^{th} epoch, since the while the loss still decreases as the network iterates over the training dataset. The evaluated loss of test sets is 1.15 with an accuracy equal to 71%.

3.2 Fast-Text Model

Then, the embedding matrix is generated for the fast text model, which is imported into the embedding layer in TensorFlow. The network is trained with batch sizes 512, 256 and 128. Since the smaller batch size would lead to a more frequent update of the model's parameters, a higher learning rate can be achieved, but it can also potentially lead to overfitting. At the same time, a small batch size would be less efficient as well. The performance of this model is displayed and the summary of results are included in a table.



Model	Batch Size	Test Accuracy	Test Loss
GloVe	256	0.710	1.148
Fast-text	128	0.736	1.219
Fast-text	256	0.717	1.305
Fast-text	512	0.654	1.379

Table 2. Summary of Results

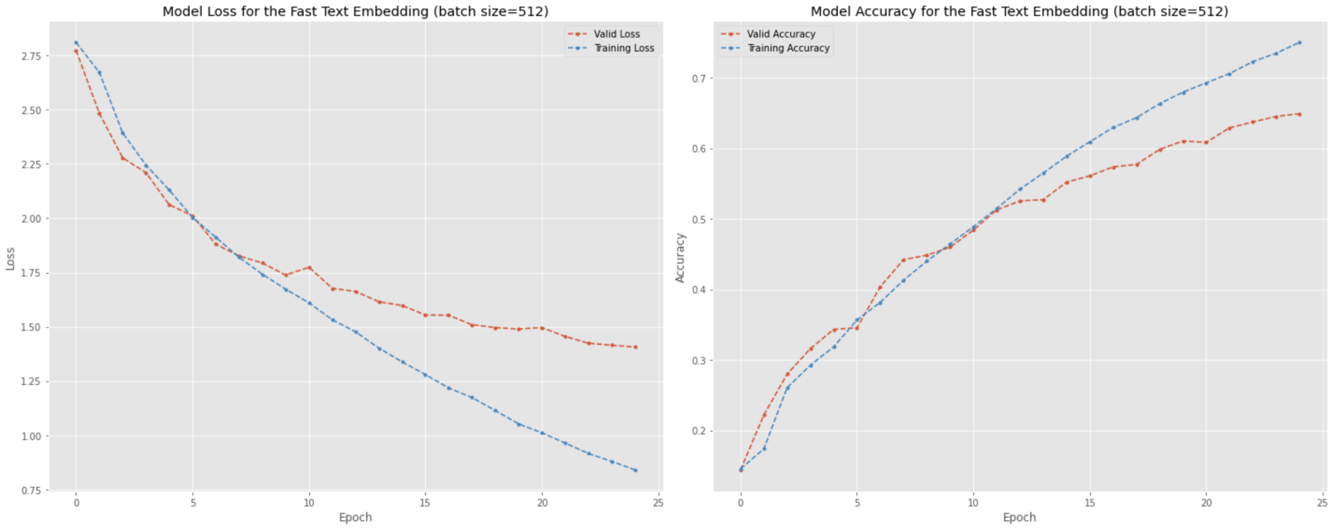


Fig. 4. Performance of the LSTM Network with Fast-Text Embedding (Batch Size=512)

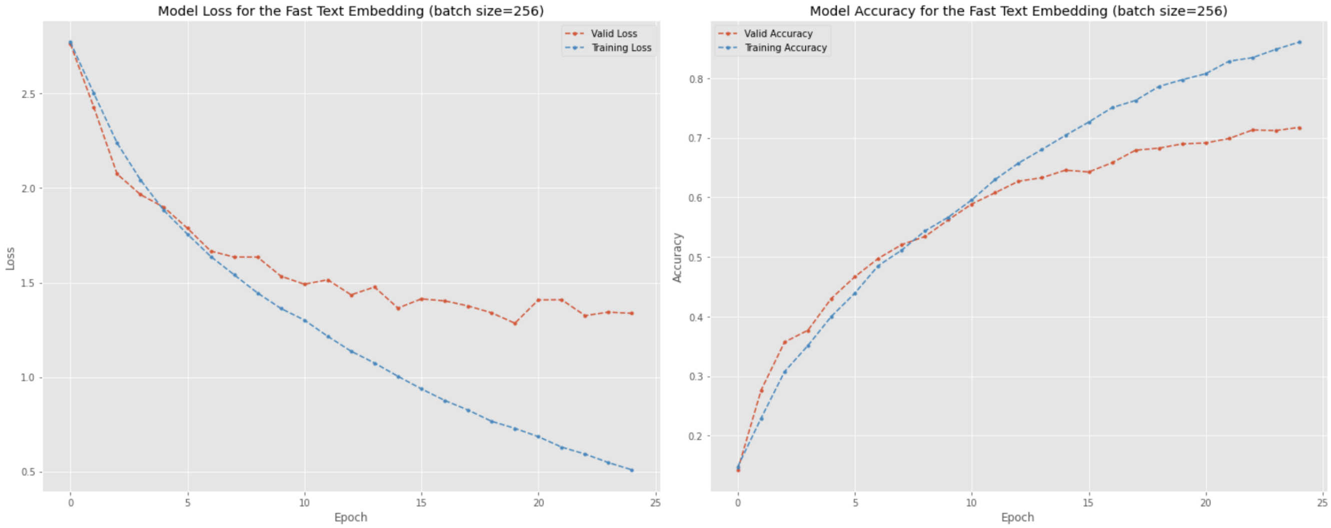


Fig. 5. Performance of the LSTM Network with Fast-Text Embedding (Batch Size=256)

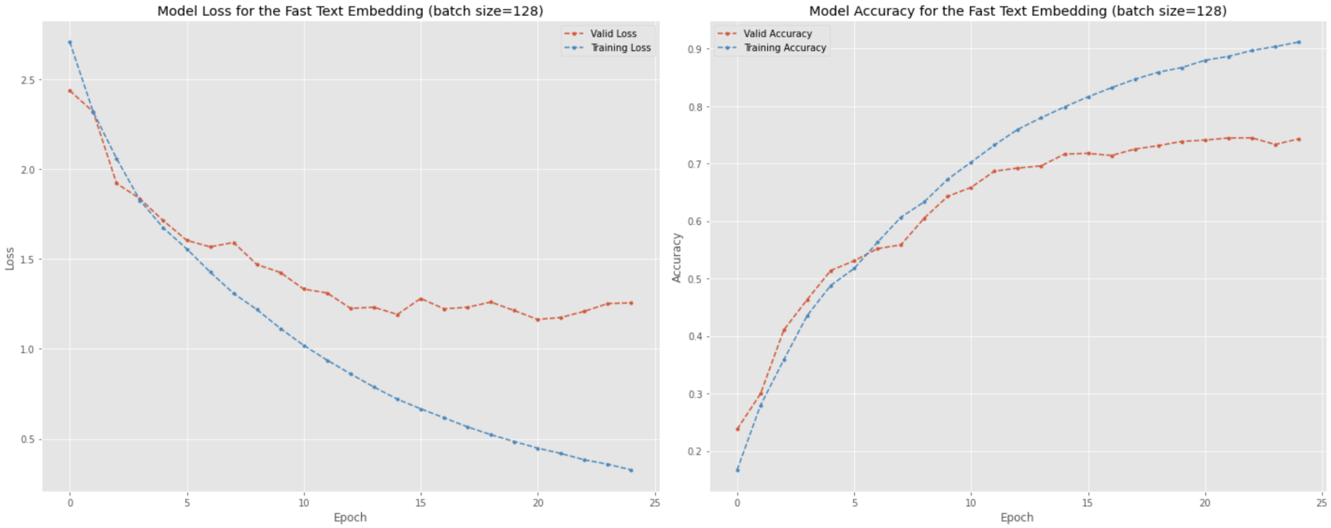


Fig. 6. Performance of the LSTM Network with Fast-Text Embedding (Batch Size=128)

When the batch size is set to 128, higher accuracy is acheived but significant overfitting also occurs while the learning rate is too low when the size is 512. Hence, the batch size of 256 would be the most suitable. Comparing two embedding models, the network with Fast-Text provided a higher accuracy while the test loss of it is actually higher than the GloVe. Therefore, for this dataset, both embedding models would produce similar results with my LSTM network, and the Fast-Text embedding is chose due to its higher accuracy and higher number of word to vector conversions.

4 DISCUSSION

In this task, the ophthalmology paper abstracts are classified into 21 MeSH terms. With the tokenisation and pre-trained Fast-Text embedding, the LSTM network reached a 74% of accuracy. There are several improvements can be made to this approach. Firstly, the K-fold cross validation can be applied to the data set, so that the data would be separated into k sets and the network can be trained and tested with different parts of data to increase its reliability. Also, since the significant overfitting is observed, other regulation techniques such as gradient clipping can be applied to prevent this effect. Furthermore, other types of networks like CNNs can be implemented to check whether better performance can be acheived.

REFERENCES

[1] Yang, P., Sun, X. and Li, W. (2018) “SGM: sequence generation model for multi-label classification.” Available at: https://doi.org/https://doi.org/10.48550/arXiv.1806.04822.

[2] Tf.keras.layers.textvectorization nbsp;: nbsp; tensorflow V2.11.0 (no date) TensorFlow. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/layers/TextVectorization (Accessed: January 10, 2023).

[3] Tf.keras.losses.CategoricalCrossentropy nbsp;: nbsp; tensorflow V2.11.0 (no date) TensorFlow. Available at: https://www.tensorflow.org/api_docs/python/tf/keras/losses/CategoricalCrossentropy (Accessed: January 12, 2023).