Tech Week '22
Machine Learning Hackathon
Method of Approach:

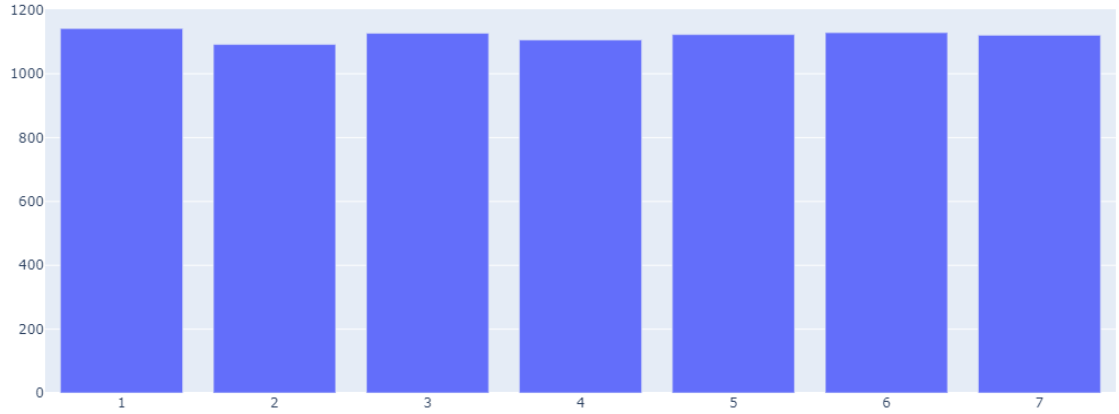
Submission by:
Team: majime
Member(s): Arunachala Amuda Murugan (2021A7PS0205H)

Part 1: Data preprocessing and exploration:

a) Our first step is to clean our text and remove unnecessary words/suffixes/prefixes:
   i) Removing stopwords, i.e, words like ‘an’, ‘the’ etc., which will not have a logical impact on the classification of text, and so, if not removed, might either cause our model to learn wrongly or introduce more computation where not required. Removing solves this problem, while also allowing our model to focus on the important information in the text.
   ii) Using common regular expressions we remove extra symbols stc. Which again would not contribute in any way if present.
   iii) We also remove any 1 letter words still there after filtering out the stopwords.
   iv) Finally we remove suffixes and prefixes by stemming the words, using the PortStemmer() class in the nltk library.

b) A problem we may face while training our model in a multi class test classification problem, is over and undersampling. This happens when one category has much more samples in the training dataset over others. But in this particular dataset, this is not the case as we see that all the labels have near equal numbers of samples:

c) The last step(s) before our data is ready for training is,



i) Tokenizing and creating vectors from our text data: We use word tokenization, provided by the Tokenizer class in tensorflow. To make all the text/token sequences/vectors of equal size we pad/truncate as required.

ii) Encoding the labels: Using OneHotEncoder in the sklearn.preprocessing

d) Finally after checking if all the types and sizes match up, we're ready to start training our model/NN

Part 2: Model/Neural Network training and tuning:

The Neural Network I have chosen for this task is a Convolutional NN.
Layer skeleton is:

1) Embedding
2) Conv1D
3) GlobalMaxPooling1D
4) Flatten
5) Dropout (to prevent overfitting)
6) Dense (Output layer)

We use the generic optimizer adam, and since this is a multi class classification problem we use the categorical_crossentropy loss function, and the activation function for the output layer is softmax.

Though RNNs are the most popular pick for NLP tasks, I have chosen to use a CNN as since they identify patterns in space, they are ideal for a classification task like the given one. (i.e., say a word/phrase which is identified as a near sure pattern to label 7, can appear in any part of our comment, our CNN can still identify it and classify it to the required class). CNNs are also much faster (~5x) times faster than an RNN, and so to train one at home, while manually filtering out hyperparameters in our own grid search made the choice of CNN much more feasible.

a) I first took a large pool of hyperparameters, and removed those that did not give even 1 ideal result in any combinations. Some parameters shortened to just one, so I fixed that, instead of putting it in the parameter grid (example: number of epochs always showed 10 in the highest few accuracies, in all combinations)
(example 2: sigmoid activation function in the Conv1D layer never gave the best results, so was removed etc.)

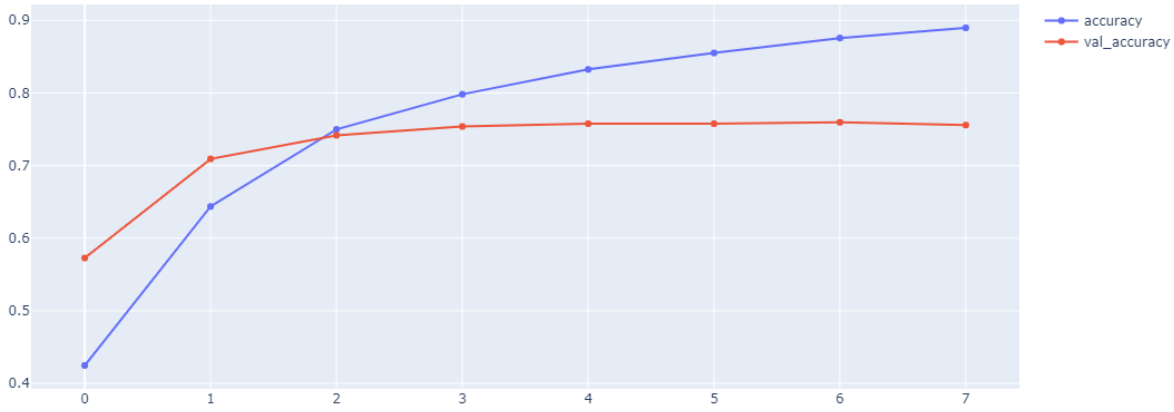
b) After reducing the pool of parameters, I ran the 'grid-search' again to find the most optimal combination of parameters. (The final grid search takes about 32 min to complete if needed to be run again, if only results are required, there is a separate csv file with them, that can be used instead of running the whole thing again)

c) In the grid search, while training our model/NN, 3 separate sets were used (from train.csv).
First we split the dataset into train and test. And we used a validation split of 0.2 on the train set while training the neural network. It does not see the test set while training. After training on the given parameters (from the parameter grid) we take a note of its accuracy of classification on this test set. Finally we choose the set of parameters which give the best accuracy.

d) Using these parameters, we train the model, but this time it is fitted on the whole dataset, not just a part, before we predict on the comments in test.csv. This is to ensure that our model has learnt everything possible from what it has seen before it tries to classify anything it hasn't seen. (To maximize accuracy)

Note: In the grid search, everytime we train the model, though we specify epochs=10, we stop earlier if the accuracy or loss on the validation set starts to plateau.

a) Accuracy plateaus an we stop at epoch 8



b) Loss plateaus and we stop at epoch 8