EX3 – 312237803

Technical assumptions

- I am using the glove 6b 100d embedding vector from torch-text, so this module should be loaded.
- I have limited time and resources, so my tuning process was enough but not optimal.
- to save running time, I saved the processed corpus in a pickle file. The code load it automatically when using the train_model() function. If you wish to run the processing you should uncomment the relevant part of the preprocess_twits_text(). Detailed explanation in the code file.
- There are 5 pickle files which are **must** for the code to run (they should be placed at the working directory): NLP_clean_text, NLP_features_df, best_model, embeddings and stoi.
- The predict() function accepts just the best model since you haven't mentioned which models it can accept, and the model inputs depend on the model. Also, the file **must** be in the same format as our testing file!

Preprocessing methodology and assumptions

- From the non-text data, I extracted one feature:

  1. part_of_day – the quarter of the day when the tweet posted from the timestamp

  columns (int (0-3)) (11 missing values were randomly assigned)

- From the text data, I extracted four features:

  1. hashtag_num – the number of hashtags in the tweet

  2. mentions_num - the number of mentions in the tweet

  3. capital_words_num – the number of capital words

  4. exclamation_mark_num – the number of exclamation marks used

  These features were standardized before they entered the models

  All can indicate the tweeting style of the writer

- I decided to remove the 'user_handle' field since it contains mostly 'RealDonaldTrump' values and will encounter a distribution shift on the test set when trump become a president. I do not want my model to rely on this feature for all 13 non ' RealDonaldTrump ' examples.
- The text data was handled by these steps to make it in similar shape as possible:

  1. URL, punctuation and digit remove

  2. Lower-case

  3. Stop-words remove

  4. Lemmatization

- Labels were converted for android and not android (0 – android (Trump), 1 – else)

The class distribution was 0 – 63% & 1 – 37%. This labels class distribution is imbalanced so my evaluation metric will consider this. I decided working with AUC score. Although the imbalance is not severe so accuracy can also be considered to my opinion. We can add some more metrics to enhance the model evaluation like precision and recall (if we decide that trump's twits are more important to classify than non-trump). To make it simple and straight forward I will stick with AUC.

Models and tuning processes

All models were evaluated using 10-fold cross validation scheme. The scores presented at each model is the mean AUC results. The first two models used the five features which I manually extracted, the FFNN and LSTM models used Glove (6B, 100D) pretrained embeddings vectors with <EOS> and padding to force all sentences be in the length.
The LSTM combined model is like the regular LSTM, but I added the five features to the linear layer before the prediction.

Model 1 – Logistic regression

This model is the first baseline for the more advanced models

Input example: [1, 5,3,4,0,0,0,1] – the first four dimensions are the text-features, the last four dimensions are four dummies for each part of day category (in the example the last part of day). I tried different C values (10 different values on a log scale between 10e-4 to 10e2, using np.logscale) to find the best parameter for this model. Graph describing the tuning process is presented below:

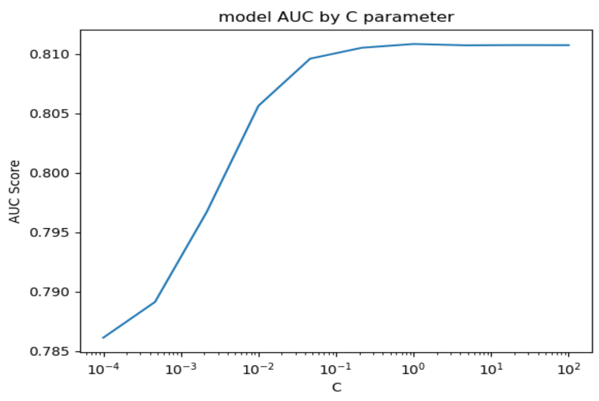

Figure 1: LR model AUC score by C parameter

The best result was using C=1 and AUC score of 81%. I will use this result as a baseline reference for the other models.

Model 2 – SVM

This model is the second baseline which also used the five manually engineered features. I tried different C values (8 different values on a log scale between 10e-4 to 10e1, using np.logscale) as well as three different kernels (sigmoid, linear and RBF) to find the best parameters for this model. Graph describing the tuning process is presented below:

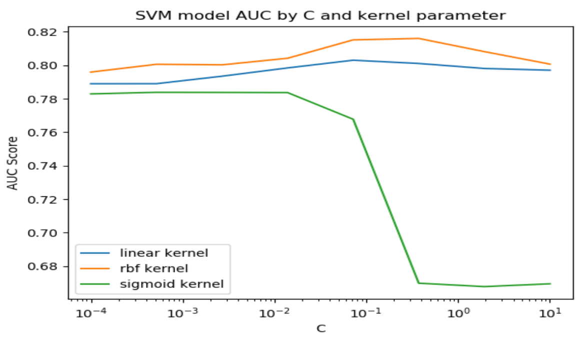

Figure 2: SVM model AUC score by C and kernel parameters

The best result was using C= 0.372 and RBF kernel with AUC score of 81.6%. the results is comparable to the LR model so I will expect the models using the embeddings vectors to outperform this baselines.

Model 3 – FFNN

This model used the Glove embeddings as input. Each input was a concatenation of all embeddings vectors to form a fixed length input as follows: max_sentence_length X 100 (the dimension of each embedding vector). As I said, I padded with <EOS> and zeros each input to form fixed length input. To tune the model, I decide to hold all parameters except the learning rate because lack of resources and training time. I think it makes the point of tuning since the learning rate is one of the most important parameters to tune in neural network.

The network has two hidden layers with 64, 32 neurons in the first and second layers. The last layer has 1 output neuron

FFNN parameters:

epochs = 10; batch size = 64; learning rates = [1e-4, 5e-4, 1e-3, 5e-3, 1e-2]

Graph describing the tuning process is presented below:

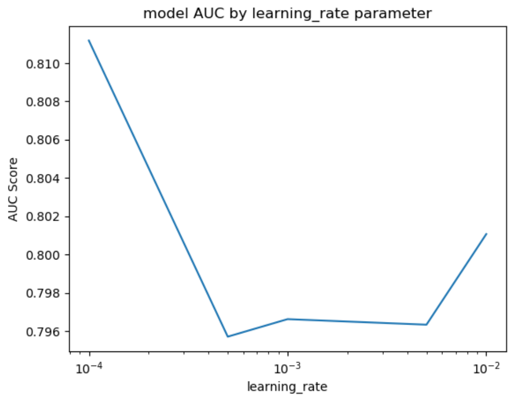

Figure 3: FFNN AUC score by learning rate parameter

The best learning rate was: 1e-4 with AUC score of 81.6%

I guess that this model didn't outperform the baseline models since the concatenation reduced some structure information of the sentences. Also, the fixed length constrain is limiting the ability to capture information from varied length sentences.

Model 4 – LSTM

This model used the same input as the FFNN model (I tried to feed variable length sentences but encountered a technical issue implementing it). The only difference was the LSTM got a sequence of tokens instead of one long input, so hopefully it will learn something about the twit length. I used shallow architecture means the LSTM had only one layer. The final output was sent to linear layer with made the classification finally. The same reasons made me tune just the learning rate parameter while the other parameters held with the following values:

LSTM parameters: epochs = 20; batch size = 64; hidden state size: 64; learning rates = [1e-4, 5e-4, 1e-3, 5e-3, 1e-2]. Graph describing the tuning process is presented below:

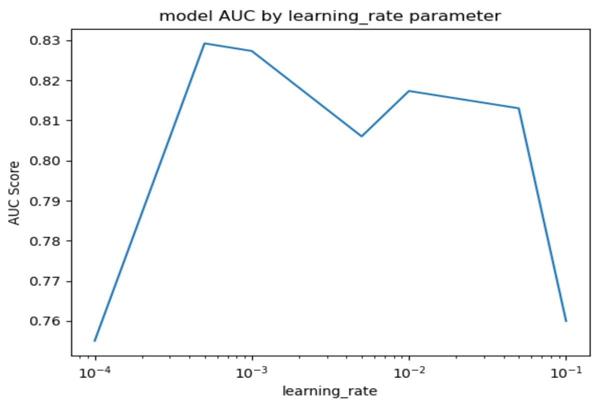

Figure 4: LSTM AUC score by learning rate parameter

The best learning rate was: 1e-4 with AUC score of 83%. Here I can see little edge above the other models. The sequence modeling may help to learn some more information. Still, I think if I could feed the network with varied length vectors the performance will improve.

Model 5 – LSTM combined with manually extracted features

The last model is the same as model 4, but here I added the five manually extracted features to the last linear layer as a concatenation to the LSTM final output. All the other things made like the LSTM model include the tuning process. Graph describing the tuning process is presented below:

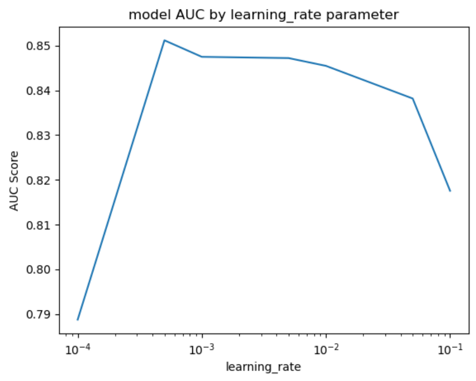

Figure 5: LSTM combined AUC score by learning rate parameter

The best learning rate was: 5e-3 with AUC score of 85.1%. this is the best model which combines two different sources of knowledge. It is suffering the same limitations as the regular LSTM.

I want to address some improvements that can be considered to improve the model. First, using different more advanced embedding vectors. Second, implementing the varied length input. Third, better domain specific (twitter) data preprocessing can be done.

Fourth, there word tokens which don't appear at Glove embeddings, hance represented as zero vectors. Handle OOV words can also improve model performance. As I said this is my best model which will be loaded using the API.