Summary Report:
Twitter Classification - Sarcasm, No Sarcasm
Steve Su (steve303) and Pierson Wodarz (wodarz2)
CS 410
12/12/2020

Description of Final Model:
In our final model we used a pre-trained Bert Model which stands for Bidirectional Encoder Representations from Transformers. It is considered a state-of-the-art model and was created and published in 2018 by Jacob Devlin and his colleagues from Google. Its performance advantage is not fully understood but one of the reasons is believed to do so well is that it takes into account the context of each word occurrence rather than treating all occurrences of the same word statically. For example, BERT will create different word vector embeddings when the word "running" is used as "running a race" versus "running a business". In contrast, other word embedding models such as Glove or word2vec assign only one vector to represent "running" in both cases. BERT (language model)

To use the BERT language model a python library called "transformer" was required. This works in conjunction with tensorflow, version 2.2. Our first step was to import the training data and convert to a pandas dataframe. Each observation (tweet) in the data frame consisted of a string. For efficiency we removed stopwords from each tweet in our preprocessing step. We also tried other preprocessing methods (removing punctuation, stopwords, non-alphanumeric characters, etc) in other models but this did not seem to provide much of an advantage. The next step was to tokenize the strings from each tweet after preprocessing. This was done using the pretrained BERT tokenizer from the transformer library. This tokenizer class also formats the data to the specific requirements, such as padding and attention mask, to be used in the BERT classification model. The next step was to create a data object from the tokenization process so that it is compatible with the BERT classification model. Now the data is ready to be trained on the BERT model. We used an Adam optimizer in the model and set the learning rate at .00005. From previous models a learning rate between .00005 and .0001 seemed to give the best results where there was a steady decrease in entropy loss. When set too high the model would overtrain too quickly and not achieve the lowest validation entropy loss. When set too low there was very little progress from epoch to epoch. To get the best score we only ran two epochs with batch size of 32. Higher epochs eroded our score ranking and increased the entropy loss. With this model we were able to beat the baseline with margin. The source code for this model is named BERT.ipynb.

credits: Fine-Tuning DistilBert for Multi-Class Text Classification using transformers and TensorFlow

Other Models:
Besides the BERT model, we also tried several other ones (available in the "Alternative Methods & Models" folder). They are listed below along with their entropy and accuracy scores in Table 1.

Model 1: Simple CNN
This model also used a BERT tokenizer but used a very simple neural network consisting of an

embedding layer (Keras) followed by a 1D convolutional layer and two dense layers. The last dense layer was the output layer with sigmoid activation function. The file name for this model is simple_model.ipynb. This model did not beat the baseline and performed the worst.

credits: Deep Convolutional Neural Network for Sentiment Analysis (Text Classification)

Model 2: Multiple Dense Layers with Dropouts

This model was similar to model 1 but two additional dense layers with dropouts were added. The dropouts randomly remove outputs between layers to prevent overfitting which should help generalization of the model. This seemed to help improve our results as it had the 3rd best score of the six models we built. However, this model did not beat the baseline. This model's filename is called dense_mode_wDropouts.ipynb.

credits: Dropout Neural Network Layer In Keras Explained | by Cory Maklin

Model 3: Model 2 + GloVe Embedding

This model tried to improve the embedding method by using GloVe instead of the one provided in the Keras library used in models one and two. GloVe is a commonly used word embedding method which has shown good results for NLP tasks. The neural network is the same as Model 2. Surprisingly, its results are worse than that of models one and two. The file name for this model is GloveTokenized_3layer_wDropouts.ipynb.

credits: How to Use Word Embedding Layers for Deep Learning with Keras

Model 4: LSTM

This is the only recurrent neural network (RNN) model we used in this project. RNNs are recognized for being able to recall information history because they employ a looping mechanism in its architecture. This seems like an advantage when handling NLP tasks, but we did not get good results with our model. There may be certain nuances that we did not understand about the model that led to poor results. Out of the six models this performed the worst. The filename for this model is lstm_model.ipynb.

Model 5: DistilBert

This model is a condensed version of the BERT model. As a result it is much faster to run compared to the full BERT model. We found the performance of these two models to be about the same. As such, this model beat the baseline.

Model	Validation Entropy Loss	Validation Accuracy
1. Simple CNN	.4934	.7660
2. Multiple dense layers w dropouts	.4626	.7760
3. Model 2 + Glove embedding	.5229	.7470
4. LSTM	.5827	.7260
5. DistillBert	.4233	.8090
6. Bert	.4155	.8090

Table 1. Performance scores of each model

Hyperparameter tuning:

Learning rate

From observations, although not documented here, we noticed that in most cases when learning rate is set to 0.01 or higher the training accuracy quickly approaches 99% training accuracy and that validation accuracy never achieves its best performance compared to setting to a smaller learning rate. The learning rate which seemed to work the best when using Adam optimizer was between 1e-4 and 1e-5. This gave a steady decrease in the validation entropy loss. However, when learning rate was set too low, less than 1e-6, validation entropy loss would not change between epochs indicating that training progress was not being made.

Number of Epochs

We found that it was important to find the optimal number of epochs during training to gain the best score. Figure 1 shows that as the number of epochs increases the training accuracy overfits and the entropy loss keeps decreasing. Overfitting starts to occur at 35 epochs. This is where the validation accuracy starts to plateau and where validation entropy approaches a minimum. So in this example we would select the training to stop at around 35 epochs. See figure 2.

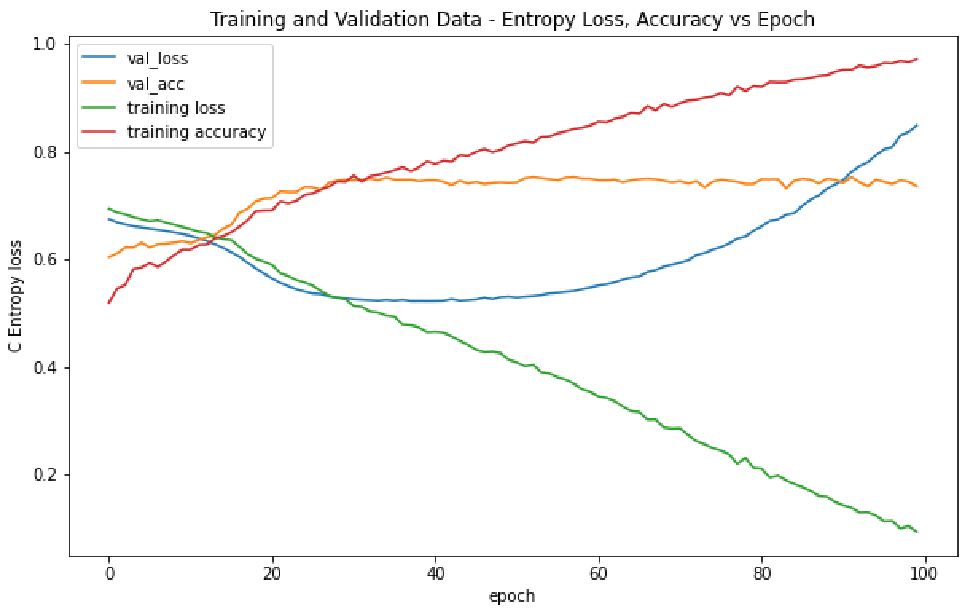

Figure 1. Training and Validation Entropy Loss, Accuracy vs. Epoch (Model 3 - GloVe Model)

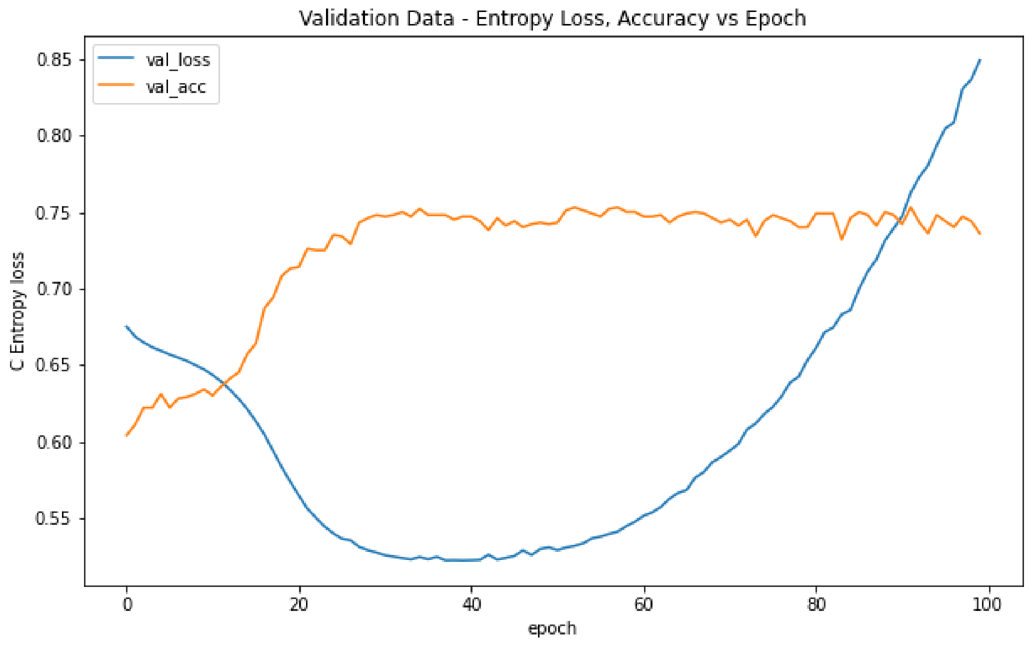

Figure 2. Validation Entropy Loss, Accuracy vs. Epoch (Model 3 - GloVe Model); Optimal number of epochs near 35

Summary:

After experimenting with several different models the DistilBert and BERT models are clearly better performers compared to our non pre-built models. None of our non pre-built models were able to beat the baseline. The best f score we achieved using a non pre-built model was about 0.70. This was model 2, the multilayer CNN with dropouts. Besides choosing the BERT model, optimizing the learning rate and number of epochs also helped our score.