Project Deliverable 3

Twitter Sentiment Analyzer

Cindy Wang
https://github.com/cindywang3299/twitter-sentiment-analyzer

1 Final Training Results

In addition to the logistic regression model combined with Tf-Idf which I trained for Deliverable 2, I spent some time training another model with Convolutional Neural Network to see whether difference in modelling can enhance the performance in prediction. In the previous model, the confusion matrix consists of positive and negative texts and predicted positive and negative texts respectively with labels between 0-1. However, due to heavily skewed data split during text preprocessing (98/1/1 ratio on training, development, and testing sets), the first model did not show a strong performance, with only 60.65% accuracy and 0.66 precision. After careful adjustment, the logistic regression model results an accuracy score of 82.44% and a 0.82 precision with a ratio of 80/10/10.

```
Accuracy Score: 60.65%
-------------------------------------------------------
Confusion Matrix

          predicted_positive  predicted_negative
positive                7094                 822
negative                5458                2586
-------------------------------------------------------
Classification Report

             precision    recall  f1-score   support

          0       0.76      0.32      0.45      8044
          1       0.57      0.90      0.69      7916

avg / total       0.66      0.61      0.57     15960
```

Figure 1: linear regression model - before adjustment

```
accuracy score: 82.44%
model is 32.04% more accurate than null accuracy
-------------------------------------------------------
Confusion Matrix

          predicted_negative  predicted_positive
negative                6503                1541
positive                1261                6655
-------------------------------------------------------
Classification Report

             precision    recall  f1-score   support

   negative       0.84      0.81      0.82      8044
   positive       0.81      0.84      0.83      7916

avg / total       0.82      0.82      0.82     15960
```

Figure 2: linear regression model - after adjustment

The CNN model with the global max pooling layer, which will extract the maximum value from each filter, yields a slightly better accuracy and precision compared to the logistic and Tf-Idf model. The output dimension will just be a 1-dimensional vector with length as same as the number of filters we applied. This can be directly passed on to a dense layer without flattening.

The best validation accuracy is from the word vectors updated through training, at epoch 3 with the validation accuracy of 82.95%. By looking at the training loss and accuracy, it seems that word embedding learned from scratch tends to overfit to the training data, and by feeding pre-trained word vectors as weights initialisation, it somewhat more generalizes and ends up having higher validation accuracy.

```
model_cnn_01 = Sequential()
e = Embedding(100000, 200, weights=[embedding_matrix], input_length=45, trainable=False)
model_cnn_01.add(e)
model_cnn_01.add(Conv1D(filters=100, kernel_size=2, padding='valid', activation='relu', strides=1))
model_cnn_01.add(GlobalMaxPooling1D())
model_cnn_01.add(Dense(256, activation='relu'))
model_cnn_01.add(Dense(1, activation='sigmoid'))
model_cnn_01.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_cnn_01.fit(x_train_seq, y_train, validation_data=(x_val_seq, y_validation), epochs=5, batch_size=32, verbose=2)
```

Figure 3: CNN1

```
model_cnn_02 = Sequential()
e = Embedding(100000, 200, input_length=45)
model_cnn_02.add(e)
model_cnn_02.add(Conv1D(filters=100, kernel_size=2, padding='valid', activation='relu', strides=1))
model_cnn_02.add(GlobalMaxPooling1D())
model_cnn_02.add(Dense(256, activation='relu'))
model_cnn_02.add(Dense(1, activation='sigmoid'))
model_cnn_02.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_cnn_02.fit(x_train_seq, y_train, validation_data=(x_val_seq, y_validation), epochs=5, batch_size=32, verbose=2)
```

Figure 4: CNN2

```
model_cnn_03 = Sequential()
e = Embedding(100000, 200, weights=[embedding_matrix], input_length=45, trainable=True)
model_cnn_03.add(e)
model_cnn_03.add(Conv1D(filters=100, kernel_size=2, padding='valid', activation='relu', strides=1))
model_cnn_03.add(GlobalMaxPooling1D())
model_cnn_03.add(Dense(256, activation='relu'))
model_cnn_03.add(Dense(1, activation='sigmoid'))
model_cnn_03.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
model_cnn_03.fit(x_train_seq, y_train, validation_data=(x_val_seq, y_validation), epochs=5, batch_size=32, verbose=2)
```

Figure 5: CNN3

2 Final demonstration proposal

I plan to integrate the model into a web app, and I decide to deploy it using Flask since it is relatively light for hosting application and keeps things simple with just Python. I will set up a landing page to host the application on a movie review classification for demo purposes using trained weights from the logistic model. The anticipated challenge in the development process is Flask deployment itself, but I will consult execs for more information if I encounter any major problems and look up resources online to make the process easier.