Ezana N. Beyenne

MSDS 453, Section 57 2020

Assignment 4: **Fake News Prediction Using Neural Network Models**

Abstract

Fake news is news deliberately spreading disinformation or hoaxes via social media or traditional print and broadcast platforms. Fake news is a medium like clickbait stories, in that they are published with the intent to sensationalize dishonest outright fabricated headlines to damage a person, entity, an agency or gain financially or politically. Steve Job's former advisor accused Facebook of failing to police misinformation because it keeps users coming back to the site (Vega, 2020). Fake news has been shown to affect people decisions and I wanted to see if we could develop a Neural Network to correctly distinguish between fake and legitimate news.

Introduction

Fake news, (also known as fabricated news), is typically found in traditional news, social media, or fake news websites, presenting news as factually accurate even though it has no basis in fact (Tufekci, 2018). Fabricated news has enormous popular appeal, are unsubstantiated stories, yet consumed by millions of people. Unfortunately, these fabrications are not only limited to politics but are also found in areas like vaccination, nutrition, and stock values. Claire Wardle identifies seven types of fake news (Wardle, 2017):

1. *Satire or parody*: material with the potential to fool, but no intention to harm.

2. *False connection*: material with visuals and/or headlines that don't match the content.

3. *Misleading content*: material intended to frame an issue or individual.

4. *False context*: material whose real content is shared with false contextual information.

5. *Impostor content*: material whose genuine sources are impersonated with false, made up stories.

6. *Manipulated content*: material with genuine information or images are doctored up to deceive.

7. *Fabricated content*: material is intended to deceive or harm with 100% fake content.

Information shapes our view of the world and allows us to make important decisions based on the information that we have. Information we receive allows us to form ideas about people or situations around us. When false, distorted, or exaggerated information is spread, we make bad decisions or form wrong opinion on such information. Fake news has led to bullying, violence against innocent people, racist ideas, fear-mongering, and it is now shown to have had a major impact on the last American presidential election (30secondes.org).

Literature review

Research using machine learning was a tool anticipated to identify fake news and prevent them from going viral and spreading misinformation (Grothaus, 2019). However, machine learning has been manipulated to easily create fake news without any human intervention, and doing a poor job of identifying fake news (Grothaus, 2019). Research conducted by two MIT doctoral students found that computers could identify machine learning generated text, but they could not identify if that text was true or false. The reason is that machine learning algorithms could easily interpret positive statements, but could not interpret negative statements (Grothaus, 2019). The database used to train machine learning algorithms called Fact Extraction and Verification (FEVER) had some inherent biases when trying to identify fake news. For machine learning algorithms to correctly identify fake news, we must somehow weed out human bias and prejudices during its training phase (Grothaus, 2019).

Kaggle also conducted a Fake News challenge, where competitors are tasked with developing a machine learning program to identify articles that might be unreliable fake news articles.

Fake news detection using various Natural Language Processing (NLP) and machine learning is still an active area of research with most of the focus being on social media platforms. The reason for this is that people have moved from traditional print and stand-alone websites to social media platforms (i.e. Twitter, Facebook, etc..) to consume news (B. Parikh, et al, 2020).

Methods

In this study, I used various neural network algorithms to see if I could correctly identify if an article was either fake news or not in the training dataset provided by the Fake News Kaggle challenge. I converted the data in train.csv into both word embedding and one hot encoding vectorization techniques to compare their effectiveness on the 1D CNN, LSTM, GRU and Bidirectional LSTM neural network methods. I then employed a tripartite splitting of the train.csv data to get a 75/15/10 percent train/dev/test split on around 18,250 rows of data after cleanup.

Table 1: Word Embedding code

```
def word_embeddding_encode_docs(tokenizer, max_length, docs):
    encoded = tokenizer.texts_to_sequences(docs) # words to integers
    padded = pad_sequences(encoded,
                           maxlen = max_length,
                           padding = 'post',
                           truncating = 'post',
                           value = 0)
    return padded
```

Table 2: One hot embedding code

```
def one_hot_encoding_docs(tokenizer, doc, max_length):
    # One hot encodode using keras
    encoded_docs = tokenizer.texts_to_matrix(doc, mode='count')
    padded = pad_sequences(encoded_docs,
                           maxlen = max_length,
                           padding = 'post',
                           truncating = 'post',
                           value = 0)
    return padded
```

Table 3: Train dev test split code

```
def train_dev_test_split(train_sequence):
    # implementing a tripartite splitting into train, dev, and test
    train_ratio = 0.75
    dev_ratio = 0.15
    test_ratio = 0.10

    # train is now 75% of the entire data set
    X_train, X_test, y_train, y_test = train_test_split(train_sequence,
                                                        y_final, test_size=(1 - train_ratio),
                                                        random_state=42)
    # test is now 10% of the initial data set
    # validation is now 15% of the initial data set
    X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test,
                                                    test_size=test_ratio/(test_ratio + dev_ratio),\
                                                   random_state=42 )
    #print(X_train.shape, X_dev.shape, X_test.shape)
    return X_train, y_train, X_dev, y_dev, X_test, y_test
```

In addition to using a factorial design with the alternative methods mentioned above using both word embedding and one hot encoding vectorization techniques, I also added alternative settings to the neural network methods by adding dropout regularizations. The dropout regularization of 25%, and 50 % on the neural networks 1D CNN, LSTM, GRU and Bidirectional LSTM where conducted on both vectorization techniques.

Early stopping was used to monitor the performance of the neural networks during the training phase in order to reduce over-fitting and improve generalization. I also was able to provide graphs of training and development set accuracy and loss, in addition to a graph of the area under the ROC curve. Since this was a binary classifcation problem (it is fake news or legitimate news), I employed the sigmoid activation function in the output layer of the neural networks, with a loss of binary crossentropy.

I initially had an area under the curve score (AUC) of 0.5, but in order to resolve that I played around with the hyperparameters of shown in table 4 and also increasing the number of hidden layers in the neural networks. I also wanted to see what the ideal number of epochs were, and with the early stopping taking place, I settled on ten epochs. The GRU netural network with one hot encoding could have used more hidden layers, since it still had an AUC score around 0.5, but I hesitated because the GRU with word embedding had a higher score.

Table 4: Hyperparameters used in the neural networks

```
# natural language processing model hyperparameters
vocab_size     = 10000          # number of unique words to use in tokenization
embedding_vector_feature = 40   # dimension of neural network embedding for a word
max_length     = 30             # number of words to be retained in each document
max_epochs     = 10
```

I used Keras Tensorflow 2.1.0 to build the four neural network functions with dropout being passed in as an optional parameter. The setup of the various neural networks is show in the tables below, this is after spending time trying to find the right combinations of layers and structures.

Table 5: Convolution 1D model

```
def Conv1D_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # define the structure of the model
    model = Sequential(name = 'conv1D_nn_model')
    model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(Conv1D(filters = 32, kernel_size = 8, activation = 'relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(80, activation = 'relu'))
    model.add(Dense(40, activation = 'relu'))
    model.add(Dense(20, activation = 'relu'))
    model.add(Dense(10, activation = 'relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

Table 6: LSTM model

```
def LSTM_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # define the structure of the model
    model = Sequential()
    model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(LSTM(100))
    model.add(Dense(80, activation = 'relu'))
    model.add(Dense(40, activation = 'relu'))
    model.add(Dense(20, activation = 'relu'))
    model.add(Dense(10, activation = 'relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

Table 7: GRU model

```
def GRU_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # GRU neural Network
    gru_model = Sequential()
    gru_model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    gru_model.add(GRU(100))
    gru_model.add(Dense(80, activation = 'relu', name = '3rd_layer'))
    gru_model.add(Dense(40, activation = 'relu', name = '4th_layer'))
    gru_model.add(Dense(20, activation = 'relu', name = '5th_layer'))
    gru_model.add(Dense(10, activation = 'relu', name = '6th_layer'))
    if dropout > 0:
        gru_model.add(tf.keras.layers.Dropout(dropout))
    gru_model.add(Dense(numberClasses, activation='sigmoid', name = 'output_layer'))
    # compiling the model
    gru_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    gru_model.summary()
    return gru_model
```

Table 8: Bidirectional LSTM

```
def Bi_RNN_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    model = Sequential()
    model.add(tf.keras.layers.Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(100)))
    model.add(tf.keras.layers.Dense(80, activation='relu'))
    model.add(tf.keras.layers.Dense(40, activation='relu'))
    model.add(tf.keras.layers.Dense(20, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

Results

The word vectorization technique that ended up with the best performance was the word embedding when compared to the one hot encoding. Additionally, from the results in Table 9, it looks like one hot encoding had the longest training execution times, but not the top test accuracy and area under the curve results. Surprisingly, the Conv1D word embedding model has the top area under the ROC curve and test accuracy scores. The results of the top neural network methods differ from the run using Jupyter notebook vs the Spyder IDE. It would be interesting to see after multiple runs, if the two editor's results begin to converge. The pdf of the AUC and graphs are shown in the *Result_Diagrams* folder and the results are stored in the *NN_Results.csv* file.

Table 9: Results of all the models in Jupyter notebook

ModelName	Training Execution Time (seconds)	Training Accuracy	Development Accuracy	Test Accuracy	Area under ROC curve
Conv1D Word embedding	14.8550	1.0000	0.9282	0.9273	0.9789
GRU.NN 0.25 Dropout Word embedding	70.2995	0.9994	0.9249	0.9262	0.9785
BidirectionLSTM 0.5 dropout Word embedding	119.2479	0.9996	0.9311	0.9267	0.9783
LSTM 0.25 Dropout Word embedding	101.1095	0.9993	0.9318	0.9278	0.9782
Conv1D 0.5 Dropout Word embedding	17.0970	0.9998	0.9249	0.9317	0.9775
Conv1D 0.25 Dropout Word embedding	14.6220	0.9999	0.9249	0.9267	0.9768
BidirectionLSTM Word embedding	131.6948	0.9993	0.9264	0.9207	0.9764
LSTM Word embedding	104.2240	0.9993	0.9304	0.9256	0.9745
GRU.NN 0.25 Dropout Word embedding	75.4989	0.9990	0.9307	0.9185	0.9742
GRU.NN Word embedding	71.4015	0.9994	0.9296	0.9273	0.9737
LSTM 0.5 Dropout Word embedding	92.8269	0.9991	0.9282	0.9196	0.9727
BidirectionLSTM 0.25 dropout Word embedding	125.5969	0.9991	0.9264	0.9169	0.9693
Bidirection LSTM One Hot Encoding	179.4108	0.9010	0.8914	0.8803	0.9347
Bidirection LSTM 0.25 Dropout One Hot Encoding	209.7008	0.9036	0.8914	0.8841	0.9342
Bidirectional LSTM 0.25 Dropout One Hot Encoding	199.2924	0.8992	0.8884	0.8797	0.9273
LSTM One Hot Encoding	166.9902	0.9033	0.8921	0.8830	0.9204
GRU.NN 0.25 Dropout One Hot Encoding	95.0537	0.8834	0.8720	0.8639	0.9102
LSTM 0.5 Dropout One Hot Encoding	147.0805	0.8901	0.8797	0.8693	0.8893
Conv1D One Hot Encoding	30.2500	0.8065	0.7969	0.8004	0.8865
Conv1D Dropout 0.25 one hot encoding	29.7790	0.8025	0.7944	0.8010	0.8797
Conv1D Dropout 0.25 one hot encoding	28.9740	0.7993	0.7907	0.7862	0.8761
LSTM 0.25 Dropout One Hot Encoding	164.2879	0.7725	0.7594	0.7704	0.8474
GRU.NN 0.5 Dropout One Hot Encoding	94.8350	0.7778	0.7634	0.7753	0.8444
GRU.NN One Hot Encoding	64.2690	0.5743	0.5840	0.5577	0.5806

Table 10: Results of all the models in Spyder notebook

ModelName	Training Execution Time (seconds)	...	Area under ROC curve
Conv1D Word embedding	16.6770	...	0.9778
Conv1D 0.25 Dropout Word embedding	17.4030	...	0.9759
BidirectionLSTM Word embedding	108.3421	...	0.9751
BidirectionLSTM 0.5 dropout Word embedding	137.1982	...	0.9748
Conv1D 0.5 Dropout Word embedding	16.1880	...	0.9746
BidirectionLSTM 0.25 dropout Word embedding	152.1701	...	0.9741
LSTM 0.25 Dropout Word embedding	102.7748	...	0.9702
LSTM Word embedding	97.6426	...	0.9698
LSTM 0.5 Dropout Word embedding	102.1575	...	0.9688
GRU.NN 0.25 Dropout Word embedding	96.6496	...	0.9683
GRU.NN Word embedding	113.4421	...	0.9679
GRU.NN 0.25 Dropout Word embedding	103.7408	...	0.9651
Bidirection LSTM One Hot Encoding	209.0252	...	0.9317
Bidirectional LSTM 0.25 Dropout One Hot Encoding	197.9242	...	0.9308
Bidirection LSTM 0.25 Dropout One Hot Encoding	202.6312	...	0.9205
LSTM 0.25 Dropout One Hot Encoding	159.9644	...	0.8899
Conv1D One Hot Encoding	23.0740	...	0.8875
Conv1D Dropout 0.25 one hot encoding	28.3782	...	0.8819
GRU.NN One Hot Encoding	132.1352	...	0.8805
GRU.NN 0.5 Dropout One Hot Encoding	126.2068	...	0.8791
Conv1D Dropout 0.25 one hot encoding	26.0612	...	0.8778
LSTM 0.5 Dropout One Hot Encoding	156.1624	...	0.8152
LSTM One Hot Encoding	160.8813	...	0.8136
GRU.NN 0.25 Dropout One Hot Encoding	110.1679	...	0.8059

Conclusions

The results of this neural network experiment do provide some really encouraging results in identifying fake or real news. It does show that word embedding along with dropout and early stopping can really create models that are good at identifying fake news, but we would have to have the social media platforms or new dissemination sites filter all the news articles through machine learning algorithms before displaying the data. In addition, they need to find a way to stop the viral spread of the fabricated news and allow the models to learn in real-time.

Fake news detection is a difficult classification technique since biases and prejudices are being introduced during training phases. Even with enough data, I do not think that we would be able to achieve high accuracy, because the language and techniques used to disseminate fabricated news are constantly evolving. It is a cat and mouse game, and the mouse seems to be winning due to their shear volume they produce.

Works Cited

Tufekci, Zeynep. (2018). It's the (Democracy-Poisoning) Golden Age of Free Speech. Retrieved from https://www.wired.com/story/free-speech-issue-tech-turmoil-new-censorship/

Vega, Nicolas. (2020) Facebook "peddling in an addictive drug called anger": Steve Jobs adviser. Retrieved from https://nypost.com/2020/06/12/facebook-peddling-in-an-addictive-drug-called-anger-steve-jobs-advisor/

Lazer, David M. J.; Baum, Matthew A.; Benkler, Yochai; Berinsky, Adam J.; Greenhill, Kelly M.; Menczer, Filippo; Metzger, Miriam J.; Nyhan, Brendan; Pennycook, Gordon. (March 9, 2018). "The science of fake news".
Retrieved from https://science.sciencemag.org/content/359/6380/1094

Wardle, Claire. (2017). Fake news. It's complicated.
Retrieved from https://firstdraftnews.org/latest/fake-news-complicated/

30secondes.org. Impacts of Fake News.
Retrieved from https://30secondes.org/en/module/impacts-of-fake-news/

Grothaus, Michael. (2019). Machine learning isn't effective at identifying fake news.
Retrieved from
https://www.fastcompany.com/90417625/machine-learning-isnt-effective-at-identifying-fake-news

Kaggle. Fake-news. Retrieved from https://www.kaggle.com/c/fake-news

B. Parikh, V. Patil and P. K. Atrey, "On the Origin, Proliferation and Tone of Fake News," *2019 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR),* San Jose, CA, USA, 2019, pp. 135-140, doi: 10.1109/MIPR.2019.00031.

Hamdi, Tarek. (2020). Top Research About Fake News Detection 2019. Retrieved from https://www.kaggle.com/c/nlp-getting-started/discussion/123454

Folder Structure:

1. **Data**: contains the train.csv
2. **Results_Diagrams**: Contains the pdf diagrams of the AUC under ROC, the Training and dev set accuracy and loss.
3. **NN_Results.csv**: contains the results from Table 9 as a csv file.
4. Jupyter notebook version of the code
5. Python version of the code

Jupyter notebook code

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import io


import tensorflow as tf
from time import time

from tensorflow.keras.layers import Dense, Dropout, Embedding, GRU, LSTM, RNN, SpatialDropout1D
from tensorflow.keras.models import Sequential
from tensorflow.keras.preprocessing.text import one_hot
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, roc_curve, auc
from sklearn.metrics import roc_auc_score
from sklearn import metrics

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Embedding
from tensorflow.keras.layers import Conv1D
from tensorflow.keras.layers import GlobalMaxPooling1D
from tensorflow.keras.utils import plot_model
from tensorflow.keras.utils import to_categorical
```

```
tf.__version__
```

```
'2.1.0'
```

```
# set up base class for callbacks to monitor training
# and for early stopping during training
#binary classification uses sigmoid and sparse_categorical_crossentropy/binary_crossentropy
tf.keras.callbacks.Callback()
```

```
<tensorflow.python.keras.callbacks.Callback at 0x1786b347b08>
```

```
earlystop_callback = \
    tf.keras.callbacks.EarlyStopping(monitor='val_accuracy',\
    min_delta=0.01, patience=5, verbose=0, mode='auto',\
    baseline=None, restore_best_weights=False)
```

```python
def plot_auc(nnName, y_test, y_pred):
    fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred)
    auc_keras = auc(fpr_keras, tpr_keras)
    plt.figure(1)
    plt.plot([0, 1], [0, 1], 'k--')
    plt.plot(fpr_keras, tpr_keras, label='Keras (area = {:.3f})'.format(auc_keras))
    plt.xlabel('False positive rate')
    plt.ylabel('True positive rate')
    plt.title('ROC curve')
    plt.legend(loc='best')
    plt.savefig('Result_Diagrams/'+ nnName + '-AUC_ZoomAUC.pdf',
        papertype = 'letter', orientation ='landscape')
    plt.show()
    plt.close()
```

```python
# The training process may be evaluated by comparing training and
# dev (validation) set performance. We use "dev" to indicate
# that these data are used in evaluating various hyperparameter
# settings. We do not test alternative hyperparameters here,
# but in other programs there will be much hyperparameter testing.
def plot_history(nnName, history):
    acc = history.history['accuracy']
    val_acc = history.history['val_accuracy']
    loss = history.history['loss']
    val_loss = history.history['val_loss']
    epoch_number = range(1, len(acc) + 1)
    plt.style.use('ggplot') # Grammar of Graphics plots
    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.plot(epoch_number, acc, 'b', label='Training')
    plt.plot(epoch_number, val_acc, 'r', label='Dev')
    plt.title('Training and Dev Set Accuracy')
    plt.xlabel('Epoch Number')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.subplot(1, 2, 2)
    plt.plot(epoch_number, loss, 'b', label='Training')
    plt.plot(epoch_number, val_loss, 'r', label='Dev')
    plt.title('Training and Dev Set Loss')
    plt.xlabel('Epoch Number')
    plt.ylabel('Loss')
    plt.legend()
    plt.savefig('Result_Diagrams/'+ nnName + 'fig-training-process.pdf',
        papertype = 'letter', orientation ='landscape')
    plt.show()
    plt.close()
```

```python
def evaluate_fitted_model_train_test(modelname, model, X_train, y_train, X_test, y_test, X_dev, y_dev):
    # evaluate fitted model on the full training set
    train_loss, train_acc = model.evaluate(X_train,y_train,verbose = 3)
    print('\n' + modelname + ' Full training set accuracy:', \
              '{:6.4f}'.format(np.round(train_acc, decimals = 4)), '\n')

    # evaluate fitted model on the full training set
    dev_loss, dev_acc = model.evaluate(X_dev,y_dev,verbose = 3)
    print( modelname + ' Development set accuracy:', \
              '{:6.4f}'.format(np.round(dev_acc, decimals = 4)), '\n')

    # evaluate the fitted model on the hold-out test set
    test_loss, test_acc = model.evaluate(X_test,  y_test, verbose = 3)
    print(modelname + ' Hold-out test set accuracy:', \
              '{:6.4f}'.format(np.round(test_acc, decimals = 4)))

    return train_acc, dev_acc, test_acc
```

```python
def evaluate_model(modelname, model, max_epochs, X_train, y_train, X_dev, y_dev, X_test, y_test, earlystop_callback):
        begin_time = time()
        history = model.fit(X_train,
                            y_train,
                            epochs = max_epochs,
                            shuffle = False,
                            validation_data = (X_dev,y_dev), verbose = 2,
                            callbacks = [earlystop_callback])
        execution_time = time() - begin_time
        print('\n' + modelname + ' Time of execution for training (seconds):', \
                 '{:10.3f}'.format(np.round(execution_time, decimals = 3)))

        #evaluate a fitted model
        train_acc, dev_acc, test_acc =\
          evaluate_fitted_model_train_test(modelname, model, X_train, y_train, X_test, y_test, X_dev, y_dev)

        y_pred_keras = model.predict(X_test)
        #print(y_pred_keras)

        # calculate roc auc
        fpr_keras, tpr_keras, thresholds_keras = roc_curve(y_test, y_pred_keras)
        roc_auc = auc(fpr_keras, tpr_keras)
        print('\n' + modelname + ' ROC AUC %.3f' % roc_auc)

        # show training process in external visualizations
        plot_history(modelname, history)

        plot_auc(modelname, y_test, y_pred_keras)


        return [modelname,execution_time, train_acc, dev_acc, test_acc,roc_auc]
```

Conv1D Model

```
def Conv1D_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # define the structure of the model
    model = Sequential(name = 'conv1D_nn_model')
    model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(Conv1D(filters = 32, kernel_size = 8, activation = 'relu'))
    model.add(GlobalMaxPooling1D())
    model.add(Dense(80, activation = 'relu'))
    model.add(Dense(40, activation = 'relu'))
    model.add(Dense(20, activation = 'relu'))
    model.add(Dense(10, activation = 'relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

LSTM Model

```
def LSTM_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # define the structure of the model
    model = Sequential()
    model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(LSTM(100))
    model.add(Dense(80, activation = 'relu'))
    model.add(Dense(40, activation = 'relu'))
    model.add(Dense(20, activation = 'relu'))
    model.add(Dense(10, activation = 'relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

GRU neural Network

```
def GRU_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    # GRU neural Network
    gru_model = Sequential()
    gru_model.add(Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    gru_model.add(GRU(100))
    gru_model.add(Dense(80, activation = 'relu', name = '3rd_layer'))
    gru_model.add(Dense(40, activation = 'relu', name = '4th_layer'))
    gru_model.add(Dense(20, activation = 'relu', name = '5th_layer'))
    gru_model.add(Dense(10, activation = 'relu', name = '6th_layer'))
    if dropout > 0:
        gru_model.add(tf.keras.layers.Dropout(dropout))
    gru_model.add(Dense(numberClasses, activation='sigmoid', name = 'output_layer'))
    # compiling the model
    gru_model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
    gru_model.summary()
    return gru_model
```

Bidirectional LSTM

```
def Bi_RNN_model(voc_size,embedding_vector_feature, max_length, numberClasses, dropout = 0):
    model = Sequential()
    model.add(tf.keras.layers.Embedding(voc_size,embedding_vector_feature, input_length=max_length))
    model.add(tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(100)))
    model.add(tf.keras.layers.Dense(80, activation='relu'))
    model.add(tf.keras.layers.Dense(40, activation='relu'))
    model.add(tf.keras.layers.Dense(20, activation='relu'))
    model.add(tf.keras.layers.Dense(10, activation='relu'))
    if dropout > 0:
        model.add(tf.keras.layers.Dropout(dropout))
    model.add(tf.keras.layers.Dense(numberClasses, activation='sigmoid'))
    model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
    model.summary()
    return model
```

Data Prep

```
train=pd.read_csv('Data/train.csv')
```

```
# determine the unique number of label classes 0/1
df_train= train.dropna()
nclasses = len(set(df_train['label'])) -1
print(nclasses)
```

```
1
```

```
#Is slightly imabalabed 4:6 ratio, no
y=df_train['label']
print(y.value_counts())
```

```
0    10361
1     7924
Name: label, dtype: int64
```

```
# natural language processing model hyperparameters
vocab_size     = 10000          # number of unique words to use in tokenization
embedding_vector_feature = 40   # dimension of neural network embedding for a word
max_length     = 30             # number of words to be retained in each document
max_epochs     = 10
result        = list()
```

Word Embedding and One Hot Encoding setup

```
df_train= train.dropna()
y= df_train['label']
y_final=np.array(y)
traindocs = df_train['title']
y_final
```

```
array([1, 0, 1, ..., 0, 1, 1], dtype=int64)
```

```
# set up tokenizer based on the training documents only
# default filter includes basic punctuation, tabs, and newlines
# filters = !"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n
# we add all numbers to this filter
# default is to convert to lowercase letters
# default is to split on spaces
# oov_token is used for out-of-vocabulary words
tokenizer = Tokenizer(num_words = vocab_size, oov_token = 'OOV',
    filters = '0123456789!"#$%&()*+,-./:;<=>?@[\]^_`{|}~\t\n')
tokenizer.fit_on_texts(traindocs)
word_index = tokenizer.word_index
# word_index is a dictionary of words and their uniquely assigned integers
print('Training vocabulary size with one out-of-vocabulary item: ',
    len(word_index))
print('\nFirst five key-value pairs in word_index:')
[print(item) for key, item in enumerate(word_index.items()) if key < 5]

# execute helper function to create a reverse dictionary
# so if we are given an index value we can retrieve the associated word
reverse_word_index = \
    dict([(value, key) for (key, value) in word_index.items()])
```

```
Training vocabulary size with one out-of-vocabulary item:  25324

First five key-value pairs in word_index:
('OOV', 1)
('the', 2)
('new', 3)
('york', 4)
('times', 5)
```

```
#https://www.onceupondata.com/2019/01/21/keras-text-part1/
def word_embeddding_encode_docs(tokenizer, max_length, docs):
    encoded = tokenizer.texts_to_sequences(docs) # words to integers
    padded = pad_sequences(encoded,
                          maxlen = max_length,
                          padding = 'post',
                          truncating = 'post',
                          value = 0)
    return padded
```

```python
#https://github.com/pemagrg1/one-hot-encoding/blob/master/one%20hot%20encoding%20using%20sklearn
def one_hot_encoding_docs(tokenizer, doc, max_length):
    # One hot encodode using keras
    encoded_docs = tokenizer.texts_to_matrix(doc, mode='count')
    padded = pad_sequences(encoded_docs,
                           maxlen = max_length,
                           padding = 'post',
                           truncating = 'post',
                           value = 0)
    return padded
```

```python
def train_dev_test_split(train_sequence):
    # implementing a tripartite splitting into train, dev, and test
    train_ratio = 0.75
    dev_ratio = 0.15
    test_ratio = 0.10

    # train is now 75% of the entire data set
    X_train, X_test, y_train, y_test = train_test_split(train_sequence,
                                                        y_final, test_size=(1 - train_ratio),
                                                        random_state=42)
    # test is now 10% of the initial data set
    # validation is now 15% of the initial data set
    X_dev, X_test, y_dev, y_test = train_test_split(X_test, y_test,
                                                    test_size=test_ratio/(test_ratio + dev_ratio),\
                                                   random_state=42 )
    #print(X_train.shape, X_dev.shape, X_test.shape)
    return X_train, y_train, X_dev, y_dev, X_test, y_test
```

```python
conv1d_model = Conv1D_model(vocab_size,embedding_vector_feature,max_length, nclasses)
```

```
Model: "conv1D_nn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_24 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
conv1d_6 (Conv1D)            (None, 23, 32)            10272     
_________________________________________________________________
global_max_pooling1d_6 (Glob (None, 32)                0         
_________________________________________________________________
dense_90 (Dense)             (None, 80)                2640      
_________________________________________________________________
dense_91 (Dense)             (None, 40)                3240      
_________________________________________________________________
dense_92 (Dense)             (None, 20)                820       
_________________________________________________________________
dense_93 (Dense)             (None, 10)                210       
_________________________________________________________________
dense_94 (Dense)             (None, 1)                 11        
=================================================================
Total params: 417,193
Trainable params: 417,193
Non-trainable params: 0
_________________________________________________________________
```

```
lstm_model = LSTM_model(vocab_size,embedding_vector_feature,max_length, nclasses)
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_25 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
lstm_12 (LSTM)               (None, 100)               56400     
_________________________________________________________________
dense_95 (Dense)             (None, 80)                8080      
_________________________________________________________________
dense_96 (Dense)             (None, 40)                3240      
_________________________________________________________________
dense_97 (Dense)             (None, 20)                820       
_________________________________________________________________
dense_98 (Dense)             (None, 10)                210       
_________________________________________________________________
dense_99 (Dense)             (None, 1)                 11        
=================================================================
Total params: 468,761
Trainable params: 468,761
Non-trainable params: 0
_________________________________________________________________
```

```
gru_model = GRU_model(vocab_size,embedding_vector_feature,max_length, nclasses)
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_26 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
gru_6 (GRU)                  (None, 100)               42600     
_________________________________________________________________
3rd_layer (Dense)            (None, 80)                8080      
_________________________________________________________________
4th_layer (Dense)            (None, 40)                3240      
_________________________________________________________________
5th_layer (Dense)            (None, 20)                820       
_________________________________________________________________
6th_layer (Dense)            (None, 10)                210       
_________________________________________________________________
output_layer (Dense)         (None, 1)                 11        
=================================================================
Total params: 454,961
Trainable params: 454,961
Non-trainable params: 0
_________________________________________________________________
```

Bidirectional RNNs

```
bi_lstm_model = Bi_RNN_model(vocab_size,embedding_vector_feature,max_length, nclasses)
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_27 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
bidirectional_6 (Bidirection (None, 200)               112800    
_________________________________________________________________
dense_100 (Dense)            (None, 80)                16080     
_________________________________________________________________
dense_101 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_102 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_103 (Dense)            (None, 10)                210       
_________________________________________________________________
dense_104 (Dense)            (None, 1)                 11        
=================================================================
Total params: 533,161
Trainable params: 533,161
Non-trainable params: 0
_________________________________________________________________
```

NN models with 0.25 Dropout

```
dropout=0.25
```

```
conv1d_model025 = Conv1D_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
lstm_model025 = LSTM_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
gru_model025 = GRU_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
bi_lstm_model025 = Bi_RNN_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
```

```
Model: "conv1D_nn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_28 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
conv1d_7 (Conv1D)            (None, 23, 32)            10272     
_________________________________________________________________
global_max_pooling1d_7 (Glob (None, 32)                0         
_________________________________________________________________
dense_105 (Dense)            (None, 80)                2640      
_________________________________________________________________
dense_106 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_107 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_108 (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_16 (Dropout)         (None, 10)                0         
_________________________________________________________________
dense_109 (Dense)            (None, 1)                 11        
=================================================================
Total params: 417,193
Trainable params: 417,193
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_21"
```

```
_________________________________________________________________
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_29 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
lstm_14 (LSTM)               (None, 100)               56400     
_________________________________________________________________
dense_110 (Dense)            (None, 80)                8080      
_________________________________________________________________
dense_111 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_112 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_113 (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_17 (Dropout)         (None, 10)                0         
_________________________________________________________________
dense_114 (Dense)            (None, 1)                 11        
=================================================================
Total params: 468,761
Trainable params: 468,761
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_22"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_30 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
gru_7 (GRU)                  (None, 100)               42600     
_________________________________________________________________
3rd_layer (Dense)            (None, 80)                8080      
_________________________________________________________________
4th_layer (Dense)            (None, 40)                3240      
_________________________________________________________________
5th_layer (Dense)            (None, 20)                820       
_________________________________________________________________
6th_layer (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_18 (Dropout)         (None, 10)                0         
_________________________________________________________________
output_layer (Dense)         (None, 1)                 11        
=================================================================
Total params: 454,961
Trainable params: 454,961
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
Model: "sequential_23"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_31 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
bidirectional_7 (Bidirection (None, 200)               112800    
_________________________________________________________________
dense_115 (Dense)            (None, 80)                16080     
_________________________________________________________________
dense_116 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_117 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_118 (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_19 (Dropout)         (None, 10)                0         
_________________________________________________________________
dense_119 (Dense)            (None, 1)                 11        
=================================================================
Total params: 533,161
Trainable params: 533,161
Non-trainable params: 0
_________________________________________________________________
```

NN models with 0.5 Dropout

```
dropout = 0.5
```

```
conv1d_model05 = Conv1D_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
lstm_model05 = LSTM_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
gru_model05 = GRU_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
bi_lstm_model05 = Bi_RNN_model(vocab_size,embedding_vector_feature,max_length, nclasses, dropout)
```

```
Model: "conv1D_nn_model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_32 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
conv1d_8 (Conv1D)            (None, 23, 32)            10272     
_________________________________________________________________
global_max_pooling1d_8 (Glob (None, 32)                0         
_________________________________________________________________
dense_120 (Dense)            (None, 80)                2640      
_________________________________________________________________
dense_121 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_122 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_123 (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_20 (Dropout)         (None, 10)                0         
_________________________________________________________________
dense_124 (Dense)            (None, 1)                 11        
=================================================================
Total params: 417,193
Trainable params: 417,193
Non-trainable params: 0
```

```
_________________________________________________________________
Model: "sequential_24"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_33 (Embedding)     (None, 30, 40)            400000
_________________________________________________________________
lstm_16 (LSTM)               (None, 100)               56400
_________________________________________________________________
dense_125 (Dense)            (None, 80)                8080
_________________________________________________________________
dense_126 (Dense)            (None, 40)                3240
_________________________________________________________________
dense_127 (Dense)            (None, 20)                820
_________________________________________________________________
dense_128 (Dense)            (None, 10)                210
_________________________________________________________________
dropout_21 (Dropout)         (None, 10)                0
_________________________________________________________________
dense_129 (Dense)            (None, 1)                 11
=================================================================
Total params: 468,761
Trainable params: 468,761
Non-trainable params: 0
_________________________________________________________________
Model: "sequential_25"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_34 (Embedding)     (None, 30, 40)            400000
_________________________________________________________________
gru_8 (GRU)                  (None, 100)               42600
_________________________________________________________________
3rd_layer (Dense)            (None, 80)                8080
_________________________________________________________________
4th_layer (Dense)            (None, 40)                3240
_________________________________________________________________
5th_layer (Dense)            (None, 20)                820
_________________________________________________________________
6th_layer (Dense)            (None, 10)                210
_________________________________________________________________
dropout_22 (Dropout)         (None, 10)                0
_________________________________________________________________
output_layer (Dense)         (None, 1)                 11
=================================================================
Total params: 454,961
Trainable params: 454,961
Non-trainable params: 0
_________________________________________________________________
```

```
_________________________________________________________________
Model: "sequential_26"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_35 (Embedding)     (None, 30, 40)            400000    
_________________________________________________________________
bidirectional_8 (Bidirection (None, 200)               112800    
_________________________________________________________________
dense_130 (Dense)            (None, 80)                16080     
_________________________________________________________________
dense_131 (Dense)            (None, 40)                3240      
_________________________________________________________________
dense_132 (Dense)            (None, 20)                820       
_________________________________________________________________
dense_133 (Dense)            (None, 10)                210       
_________________________________________________________________
dropout_23 (Dropout)         (None, 10)                0         
_________________________________________________________________
dense_134 (Dense)            (None, 1)                 11        
=================================================================
Total params: 533,161
Trainable params: 533,161
Non-trainable params: 0
_________________________________________________________________
```

Generally speaking, an experiment comparing the performance of alternative neural network language models or techniques would be a good research topic for this assignment.

- 1. Consider using a factorial design with alternative methods (dense, 1D CNN, LSTM, versus GRU) as one of the factors.
- 2. A comparison of one-hot encoding versus word embeddings for word/term vectorization could be another factor.
- Comparisons across alternative settings for dropout regularization (none, 5 percent,50 percent) may also be useful.

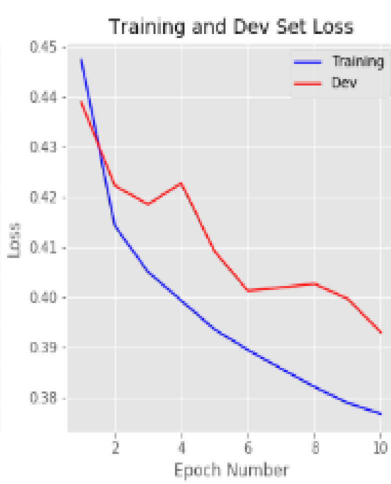
One hot encoding

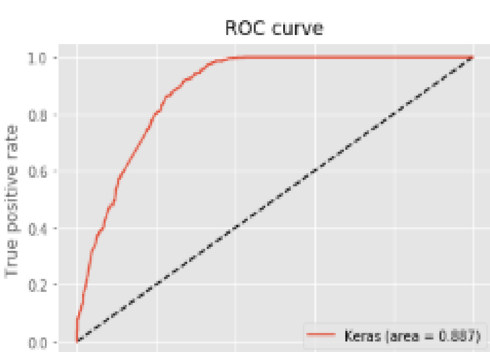
```
train_sequence_one_hotEncoding = one_hot_encoding_docs(tokenizer, traindocs, max_length)
train_sequence_one_hotEncoding

#One hot encoding
X_train, y_train, X_dev, y_dev, X_test, y_test = train_dev_test_split(train_sequence_one_hotEncoding)
```

Conv1D Model

```
res = evaluate_model('Conv1D One Hot Encoding', conv1d_model, max_epochs, X_train, y_train, X_dev, y_dev,
                                                   X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 3s - loss: 0.4472 - accuracy: 0.7642 - val_loss: 0.4388 - val_accuracy: 0.7649
Epoch 2/10
13713/13713 - 3s - loss: 0.4142 - accuracy: 0.7815 - val_loss: 0.4222 - val_accuracy: 0.7714
Epoch 3/10
13713/13713 - 3s - loss: 0.4050 - accuracy: 0.7866 - val_loss: 0.4184 - val_accuracy: 0.7743
Epoch 4/10
13713/13713 - 3s - loss: 0.3993 - accuracy: 0.7903 - val_loss: 0.4227 - val_accuracy: 0.7773
Epoch 5/10
13713/13713 - 3s - loss: 0.3935 - accuracy: 0.7922 - val_loss: 0.4091 - val_accuracy: 0.7849
Epoch 6/10
13713/13713 - 3s - loss: 0.3894 - accuracy: 0.7962 - val_loss: 0.4012 - val_accuracy: 0.7860
Epoch 7/10
13713/13713 - 3s - loss: 0.3857 - accuracy: 0.7985 - val_loss: 0.4018 - val_accuracy: 0.7882
Epoch 8/10
13713/13713 - 3s - loss: 0.3821 - accuracy: 0.8000 - val_loss: 0.4026 - val_accuracy: 0.7849
Epoch 9/10
13713/13713 - 3s - loss: 0.3788 - accuracy: 0.8036 - val_loss: 0.3996 - val_accuracy: 0.7951
Epoch 10/10
13713/13713 - 3s - loss: 0.3766 - accuracy: 0.8050 - val_loss: 0.3929 - val_accuracy: 0.7969

Conv1D One Hot Encoding Time of execution for training (seconds):     30.250

Conv1D One Hot Encoding Full training set accuracy: 0.8065

Conv1D One Hot Encoding Development set accuracy: 0.7969

Conv1D One Hot Encoding Hold-out test set accuracy: 0.8004

Conv1D One Hot Encoding ROC AUC 0.887
```

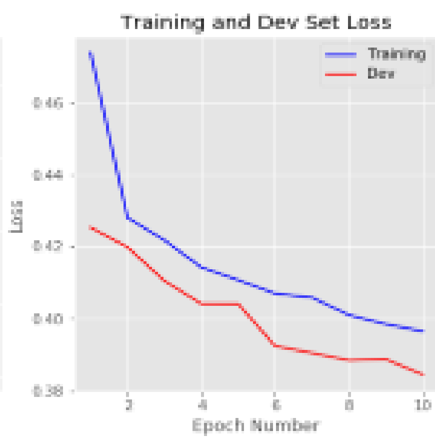


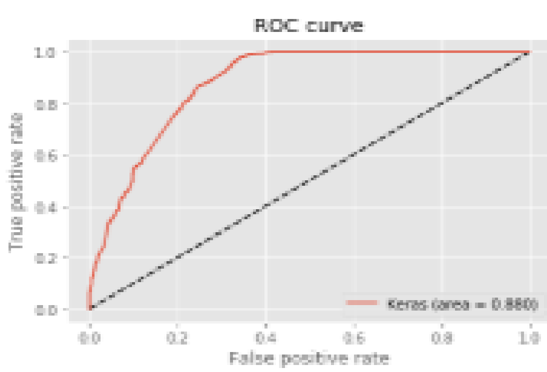


Conv1D Model 0.25

```
res = evaluate_model('Conv1D Dropout 0.25 one hot encoding', conv1d_model025,
                                     max_epochs,
                                     X_train, y_train,
                                     X_dev, y_dev,
                                     X_test, y_test,
                                     earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 3s - loss: 0.4739 - accuracy: 0.7503 - val_loss: 0.4254 - val_accuracy: 0.7681
Epoch 2/10
13713/13713 - 3s - loss: 0.4281 - accuracy: 0.7744 - val_loss: 0.4199 - val_accuracy: 0.7685
Epoch 3/10
13713/13713 - 3s - loss: 0.4218 - accuracy: 0.7754 - val_loss: 0.4105 - val_accuracy: 0.7754
Epoch 4/10
13713/13713 - 3s - loss: 0.4143 - accuracy: 0.7800 - val_loss: 0.4041 - val_accuracy: 0.7773
Epoch 5/10
13713/13713 - 3s - loss: 0.4107 - accuracy: 0.7834 - val_loss: 0.4041 - val_accuracy: 0.7802
Epoch 6/10
13713/13713 - 3s - loss: 0.4070 - accuracy: 0.7846 - val_loss: 0.3924 - val_accuracy: 0.7918
Epoch 7/10
13713/13713 - 3s - loss: 0.4059 - accuracy: 0.7894 - val_loss: 0.3905 - val_accuracy: 0.7889
Epoch 8/10
13713/13713 - 3s - loss: 0.4010 - accuracy: 0.7893 - val_loss: 0.3884 - val_accuracy: 0.7933
Epoch 9/10
13713/13713 - 3s - loss: 0.3985 - accuracy: 0.7910 - val_loss: 0.3887 - val_accuracy: 0.7944
Epoch 10/10
13713/13713 - 3s - loss: 0.3965 - accuracy: 0.7922 - val_loss: 0.3844 - val_accuracy: 0.7944

Conv1D Dropout 0.25 one hot encoding Time of execution for training (seconds):     29.779

Conv1D Dropout 0.25 one hot encoding Full training set accuracy: 0.8025

Conv1D Dropout 0.25 one hot encoding Development set accuracy: 0.7944

Conv1D Dropout 0.25 one hot encoding Hold-out test set accuracy: 0.8010

Conv1D Dropout 0.25 one hot encoding ROC AUC 0.880
```

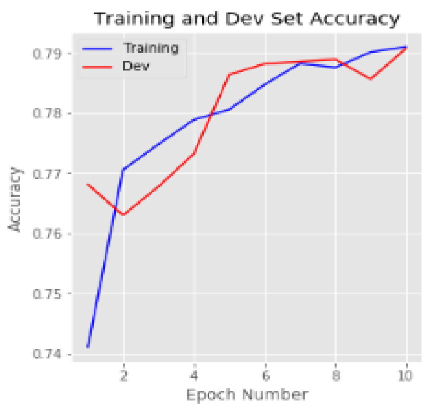


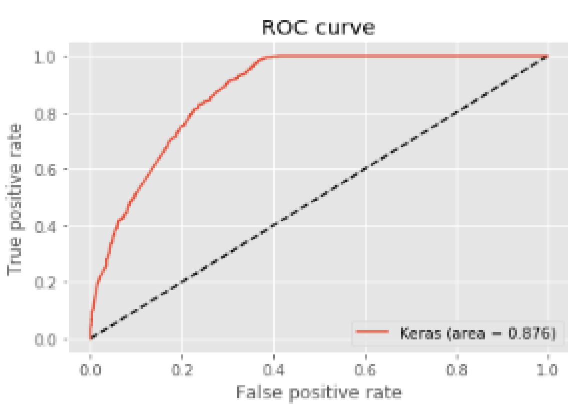


Conv1D Model 0.5

```
res = evaluate_model('Conv1D Dropout 0.25 one hot encoding', conv1d_model05,
                                        max_epochs,
                                        X_train, y_train,
                                        X_dev, y_dev,
                                        X_test, y_test,
                                        earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 3s - loss: 0.4797 - accuracy: 0.7411 - val_loss: 0.4323 - val_accuracy: 0.7681
Epoch 2/10
13713/13713 - 3s - loss: 0.4371 - accuracy: 0.7706 - val_loss: 0.4252 - val_accuracy: 0.7630
Epoch 3/10
13713/13713 - 3s - loss: 0.4270 - accuracy: 0.7748 - val_loss: 0.4112 - val_accuracy: 0.7678
Epoch 4/10
13713/13713 - 3s - loss: 0.4175 - accuracy: 0.7789 - val_loss: 0.4106 - val_accuracy: 0.7732
Epoch 5/10
13713/13713 - 3s - loss: 0.4162 - accuracy: 0.7806 - val_loss: 0.3996 - val_accuracy: 0.7864
Epoch 6/10
13713/13713 - 3s - loss: 0.4140 - accuracy: 0.7847 - val_loss: 0.3975 - val_accuracy: 0.7882
Epoch 7/10
13713/13713 - 3s - loss: 0.4092 - accuracy: 0.7882 - val_loss: 0.3940 - val_accuracy: 0.7886
Epoch 8/10
13713/13713 - 3s - loss: 0.4084 - accuracy: 0.7876 - val_loss: 0.3951 - val_accuracy: 0.7889
Epoch 9/10
13713/13713 - 3s - loss: 0.4053 - accuracy: 0.7901 - val_loss: 0.3956 - val_accuracy: 0.7856
Epoch 10/10
13713/13713 - 3s - loss: 0.4078 - accuracy: 0.7910 - val_loss: 0.3910 - val_accuracy: 0.7907

Conv1D Dropout 0.25 one hot encoding Time of execution for training (seconds):     28.974

Conv1D Dropout 0.25 one hot encoding Full training set accuracy: 0.7993

Conv1D Dropout 0.25 one hot encoding Development set accuracy: 0.7907

Conv1D Dropout 0.25 one hot encoding Hold-out test set accuracy: 0.7862

Conv1D Dropout 0.25 one hot encoding ROC AUC 0.876
```

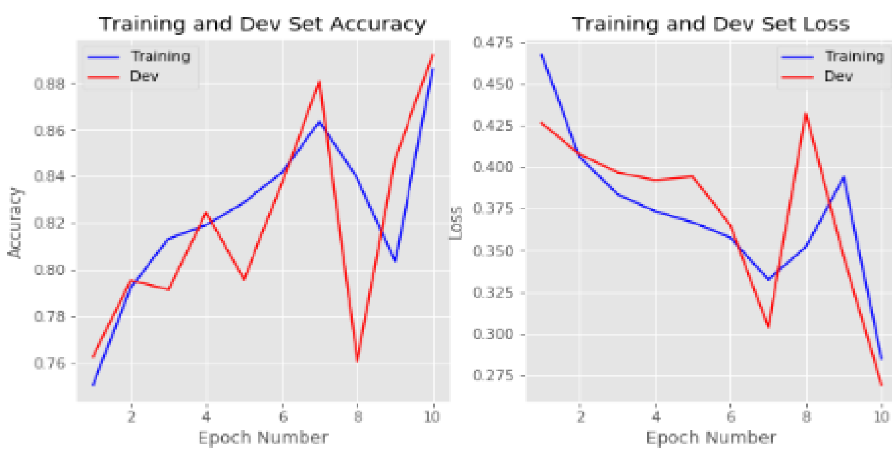


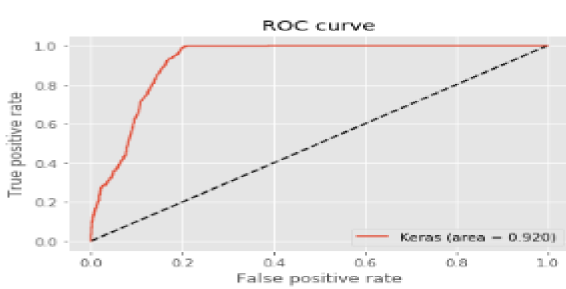


LSTM Model

```
res =evaluate_model('LSTM One Hot Encoding', lstm_model,
                                              max_epochs,
                                              X_train, y_train,
                                              X_dev, y_dev,
                                              X_test, y_test,
                                              earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 17s - loss: 0.4673 - accuracy: 0.7505 - val_loss: 0.4263 - val_accuracy: 0.7627
Epoch 2/10
13713/13713 - 16s - loss: 0.4062 - accuracy: 0.7923 - val_loss: 0.4078 - val_accuracy: 0.7955
Epoch 3/10
13713/13713 - 17s - loss: 0.3837 - accuracy: 0.8133 - val_loss: 0.3969 - val_accuracy: 0.7915
Epoch 4/10
13713/13713 - 16s - loss: 0.3734 - accuracy: 0.8191 - val_loss: 0.3921 - val_accuracy: 0.8246
Epoch 5/10
13713/13713 - 17s - loss: 0.3667 - accuracy: 0.8288 - val_loss: 0.3942 - val_accuracy: 0.7958
Epoch 6/10
13713/13713 - 16s - loss: 0.3576 - accuracy: 0.8417 - val_loss: 0.3649 - val_accuracy: 0.8370
Epoch 7/10
13713/13713 - 17s - loss: 0.3323 - accuracy: 0.8635 - val_loss: 0.3036 - val_accuracy: 0.8808
Epoch 8/10
13713/13713 - 16s - loss: 0.3520 - accuracy: 0.8391 - val_loss: 0.4322 - val_accuracy: 0.7608
Epoch 9/10
13713/13713 - 17s - loss: 0.3938 - accuracy: 0.8036 - val_loss: 0.3460 - val_accuracy: 0.8476
Epoch 10/10
13713/13713 - 17s - loss: 0.2847 - accuracy: 0.8860 - val_loss: 0.2688 - val_accuracy: 0.8921

LSTM One Hot Encoding Time of execution for training (seconds):    166.990

LSTM One Hot Encoding Full training set accuracy: 0.9033

LSTM One Hot Encoding Development set accuracy: 0.8921

LSTM One Hot Encoding Hold-out test set accuracy: 0.8830

LSTM One Hot Encoding ROC AUC 0.920
```

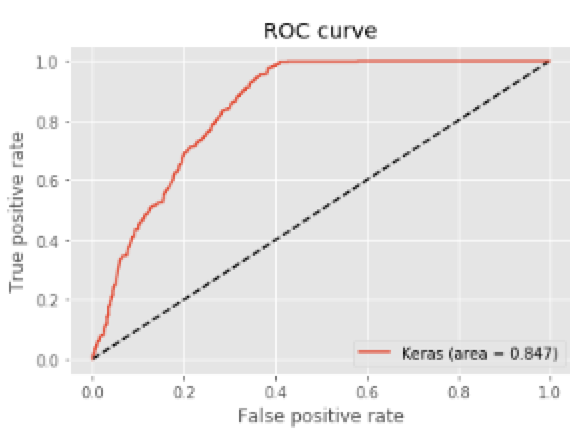




LSTM Model Dropout 0.25

```
res =evaluate_model('LSTM 0.25 Dropout One Hot Encoding', lstm_model025,
                                               max_epochs,
                                               X_train, y_train,
                                               X_dev, y_dev,
                                               X_test, y_test,
                                               earlystop_callback)
result.append(res)
```

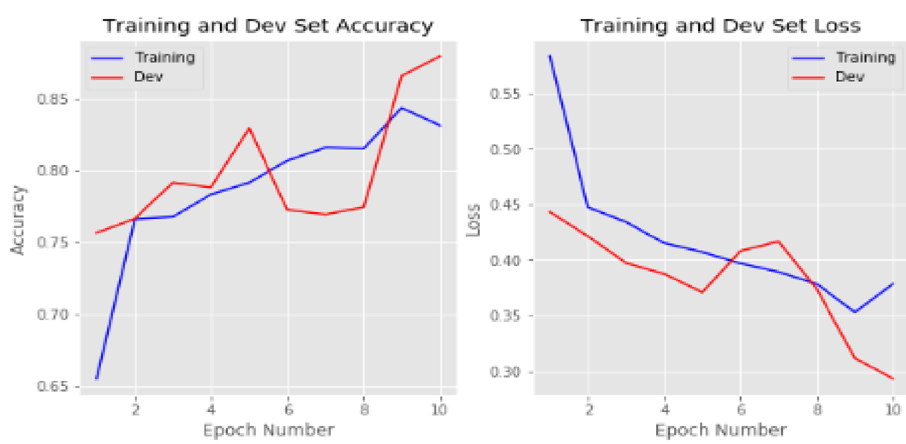
```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 18s - loss: 0.5108 - accuracy: 0.7300 - val_loss: 0.4622 - val_accuracy: 0.7539
Epoch 2/10
13713/13713 - 16s - loss: 0.4389 - accuracy: 0.7727 - val_loss: 0.4237 - val_accuracy: 0.7692
Epoch 3/10
13713/13713 - 16s - loss: 0.4060 - accuracy: 0.7959 - val_loss: 0.4094 - val_accuracy: 0.7886
Epoch 4/10
13713/13713 - 16s - loss: 0.3895 - accuracy: 0.8062 - val_loss: 0.3999 - val_accuracy: 0.7988
Epoch 5/10
13713/13713 - 16s - loss: 0.3931 - accuracy: 0.8053 - val_loss: 0.3987 - val_accuracy: 0.7758
Epoch 6/10
13713/13713 - 16s - loss: 0.3831 - accuracy: 0.8165 - val_loss: 0.3817 - val_accuracy: 0.8068
Epoch 7/10
13713/13713 - 17s - loss: 0.3847 - accuracy: 0.8205 - val_loss: 0.3884 - val_accuracy: 0.8050
Epoch 8/10
13713/13713 - 17s - loss: 0.3759 - accuracy: 0.8259 - val_loss: 0.3585 - val_accuracy: 0.8392
Epoch 9/10
13713/13713 - 16s - loss: 0.3746 - accuracy: 0.8283 - val_loss: 0.3485 - val_accuracy: 0.8531
Epoch 10/10
13713/13713 - 16s - loss: 0.3921 - accuracy: 0.8179 - val_loss: 0.4340 - val_accuracy: 0.7594

LSTM 0.25 Dropout One Hot Encoding Time of execution for training (seconds):    164.288

LSTM 0.25 Dropout One Hot Encoding Full training set accuracy: 0.7725

LSTM 0.25 Dropout One Hot Encoding Development set accuracy: 0.7594

LSTM 0.25 Dropout One Hot Encoding Hold-out test set accuracy: 0.7704

LSTM 0.25 Dropout One Hot Encoding ROC AUC 0.847
```

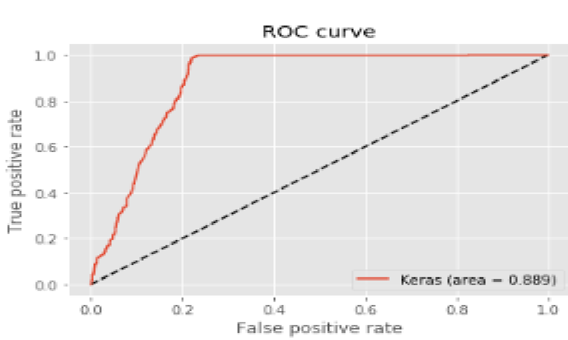


LSTM Model Dropout 0.5

```
res =evaluate_model('LSTM 0.5 Dropout One Hot Encoding', lstm_model05,
                                              max_epochs,
                                              X_train, y_train,
                                              X_dev, y_dev,
                                              X_test, y_test,
                                              earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 17s - loss: 0.5834 - accuracy: 0.6546 - val_loss: 0.4432 - val_accuracy: 0.7565
Epoch 2/10
13713/13713 - 15s - loss: 0.4474 - accuracy: 0.7661 - val_loss: 0.4214 - val_accuracy: 0.7663
Epoch 3/10
13713/13713 - 15s - loss: 0.4341 - accuracy: 0.7676 - val_loss: 0.3972 - val_accuracy: 0.7915
Epoch 4/10
13713/13713 - 13s - loss: 0.4152 - accuracy: 0.7833 - val_loss: 0.3875 - val_accuracy: 0.7882
Epoch 5/10
13713/13713 - 14s - loss: 0.4071 - accuracy: 0.7915 - val_loss: 0.3707 - val_accuracy: 0.8294
Epoch 6/10
13713/13713 - 15s - loss: 0.3969 - accuracy: 0.8068 - val_loss: 0.4081 - val_accuracy: 0.7725
Epoch 7/10
13713/13713 - 14s - loss: 0.3893 - accuracy: 0.8160 - val_loss: 0.4166 - val_accuracy: 0.7692
Epoch 8/10
13713/13713 - 14s - loss: 0.3784 - accuracy: 0.8152 - val_loss: 0.3738 - val_accuracy: 0.7743
Epoch 9/10
13713/13713 - 14s - loss: 0.3531 - accuracy: 0.8435 - val_loss: 0.3117 - val_accuracy: 0.8658
Epoch 10/10
13713/13713 - 14s - loss: 0.3786 - accuracy: 0.8313 - val_loss: 0.2931 - val_accuracy: 0.8797

LSTM 0.5 Dropout One Hot Encoding Time of execution for training (seconds):    147.080

LSTM 0.5 Dropout One Hot Encoding Full training set accuracy: 0.8901

LSTM 0.5 Dropout One Hot Encoding Development set accuracy: 0.8797

LSTM 0.5 Dropout One Hot Encoding Hold-out test set accuracy: 0.8693

LSTM 0.5 Dropout One Hot Encoding ROC AUC 0.889
```

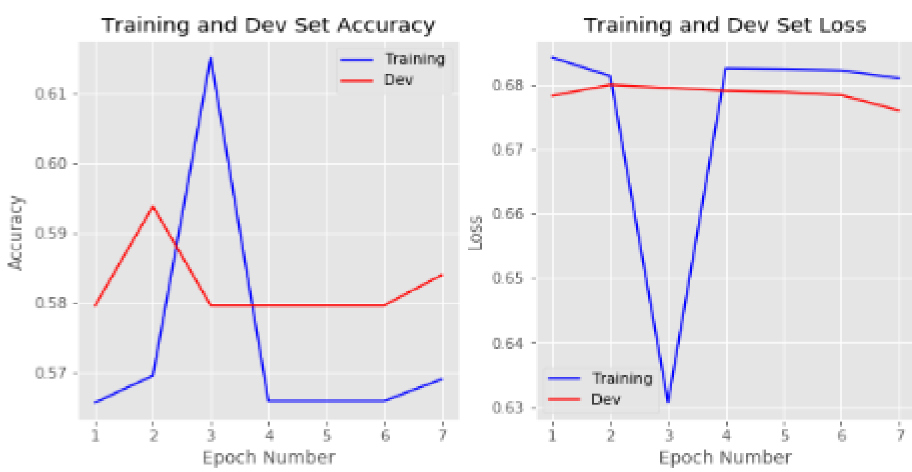


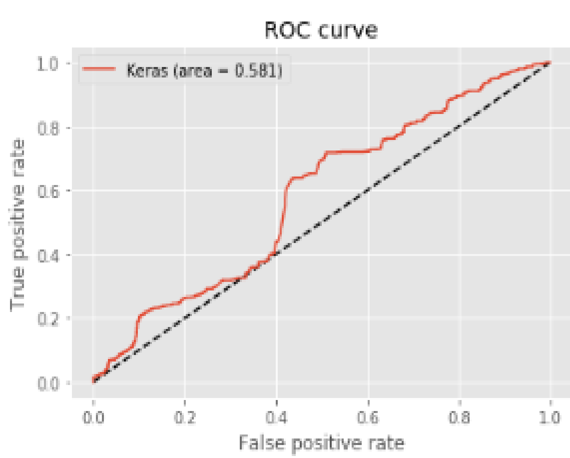


GRU Model

```
res = evaluate_model('GRU.NN One Hot Encoding', gru_model, max_epochs, X_train, y_train,
                     X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 11s - loss: 0.6842 - accuracy: 0.5657 - val_loss: 0.6783 - val_accuracy: 0.5797
Epoch 2/10
13713/13713 - 10s - loss: 0.6813 - accuracy: 0.5696 - val_loss: 0.6800 - val_accuracy: 0.5939
Epoch 3/10
13713/13713 - 9s - loss: 0.6307 - accuracy: 0.6152 - val_loss: 0.6795 - val_accuracy: 0.5797
Epoch 4/10
13713/13713 - 9s - loss: 0.6825 - accuracy: 0.5660 - val_loss: 0.6791 - val_accuracy: 0.5797
Epoch 5/10
13713/13713 - 9s - loss: 0.6824 - accuracy: 0.5660 - val_loss: 0.6789 - val_accuracy: 0.5797
Epoch 6/10
13713/13713 - 8s - loss: 0.6822 - accuracy: 0.5660 - val_loss: 0.6784 - val_accuracy: 0.5797
Epoch 7/10
13713/13713 - 9s - loss: 0.6810 - accuracy: 0.5691 - val_loss: 0.6760 - val_accuracy: 0.5840

GRU.NN One Hot Encoding Time of execution for training (seconds):     64.269

GRU.NN One Hot Encoding Full training set accuracy: 0.5743

GRU.NN One Hot Encoding Development set accuracy: 0.5840

GRU.NN One Hot Encoding Hold-out test set accuracy: 0.5577

GRU.NN One Hot Encoding ROC AUC 0.581
```

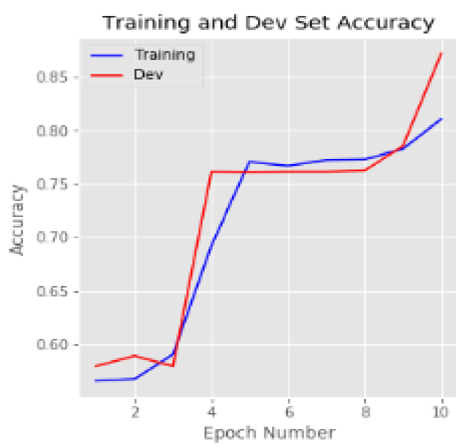


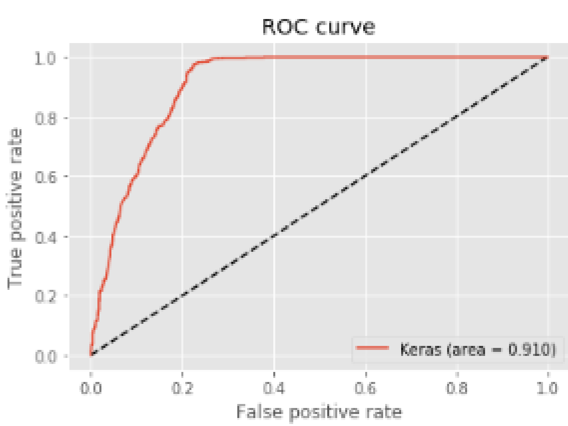


GRU Model 0.25

```
res = evaluate_model('GRU.NN 0.25 Dropout One Hot Encoding', gru_model025, max_epochs, X_train, y_train,
                           X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 12s - loss: 0.6852 - accuracy: 0.5659 - val_loss: 0.6785 - val_accuracy: 0.5797
Epoch 2/10
13713/13713 - 9s - loss: 0.6811 - accuracy: 0.5675 - val_loss: 0.6398 - val_accuracy: 0.5891
Epoch 3/10
13713/13713 - 9s - loss: 0.6614 - accuracy: 0.5906 - val_loss: 0.6838 - val_accuracy: 0.5797
Epoch 4/10
13713/13713 - 9s - loss: 0.5323 - accuracy: 0.6924 - val_loss: 0.4350 - val_accuracy: 0.7616
Epoch 5/10
13713/13713 - 9s - loss: 0.4351 - accuracy: 0.7707 - val_loss: 0.4361 - val_accuracy: 0.7612
Epoch 6/10
13713/13713 - 9s - loss: 0.4394 - accuracy: 0.7669 - val_loss: 0.4343 - val_accuracy: 0.7616
Epoch 7/10
13713/13713 - 9s - loss: 0.4339 - accuracy: 0.7724 - val_loss: 0.4308 - val_accuracy: 0.7616
Epoch 8/10
13713/13713 - 10s - loss: 0.4268 - accuracy: 0.7730 - val_loss: 0.4346 - val_accuracy: 0.7627
Epoch 9/10
13713/13713 - 10s - loss: 0.4129 - accuracy: 0.7827 - val_loss: 0.4050 - val_accuracy: 0.7853
Epoch 10/10
13713/13713 - 9s - loss: 0.3893 - accuracy: 0.8107 - val_loss: 0.3140 - val_accuracy: 0.8720

GRU.NN 0.25 Dropout One Hot Encoding Time of execution for training (seconds):     95.054

GRU.NN 0.25 Dropout One Hot Encoding Full training set accuracy: 0.8834

GRU.NN 0.25 Dropout One Hot Encoding Development set accuracy: 0.8720

GRU.NN 0.25 Dropout One Hot Encoding Hold-out test set accuracy: 0.8639

GRU.NN 0.25 Dropout One Hot Encoding ROC AUC 0.910
```

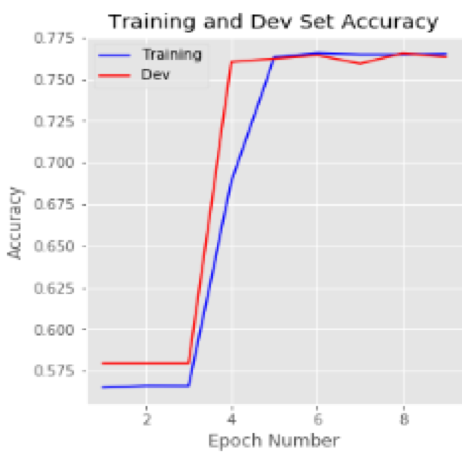


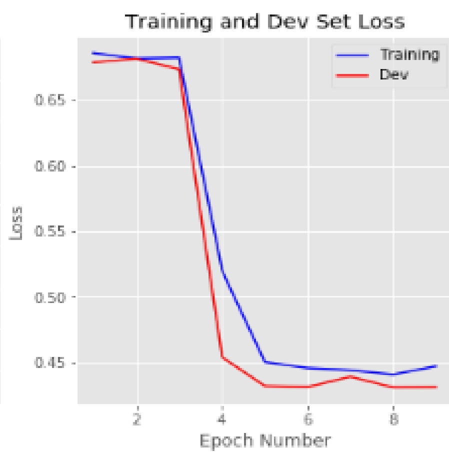


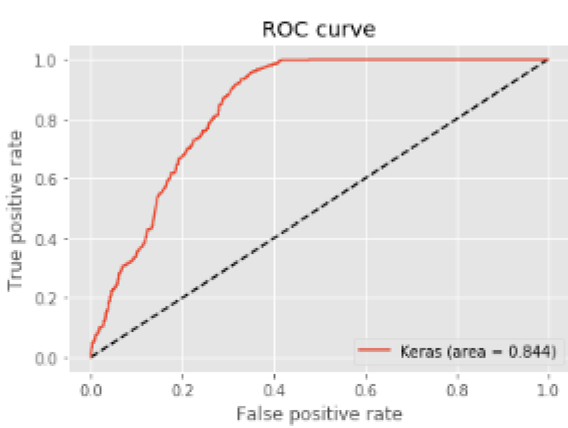
GRU Model 0.5

```
res = evaluate_model('GRU.NN 0.5 Dropout One Hot Encoding', gru_model05, max_epochs, X_train, y_train,
                     X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 13s - loss: 0.6858 - accuracy: 0.5651 - val_loss: 0.6788 - val_accuracy: 0.5797
Epoch 2/10
13713/13713 - 11s - loss: 0.6817 - accuracy: 0.5660 - val_loss: 0.6816 - val_accuracy: 0.5797
Epoch 3/10
13713/13713 - 10s - loss: 0.6824 - accuracy: 0.5660 - val_loss: 0.6736 - val_accuracy: 0.5797
Epoch 4/10
13713/13713 - 10s - loss: 0.5205 - accuracy: 0.6887 - val_loss: 0.4541 - val_accuracy: 0.7605
Epoch 5/10
13713/13713 - 10s - loss: 0.4502 - accuracy: 0.7633 - val_loss: 0.4319 - val_accuracy: 0.7623
Epoch 6/10
13713/13713 - 10s - loss: 0.4456 - accuracy: 0.7657 - val_loss: 0.4312 - val_accuracy: 0.7645
Epoch 7/10
13713/13713 - 10s - loss: 0.4440 - accuracy: 0.7648 - val_loss: 0.4390 - val_accuracy: 0.7594
Epoch 8/10
13713/13713 - 10s - loss: 0.4407 - accuracy: 0.7648 - val_loss: 0.4308 - val_accuracy: 0.7656
Epoch 9/10
13713/13713 - 10s - loss: 0.4471 - accuracy: 0.7652 - val_loss: 0.4309 - val_accuracy: 0.7634

GRU.NN 0.5 Dropout One Hot Encoding Time of execution for training (seconds):     94.835

GRU.NN 0.5 Dropout One Hot Encoding Full training set accuracy: 0.7778

GRU.NN 0.5 Dropout One Hot Encoding Development set accuracy: 0.7634

GRU.NN 0.5 Dropout One Hot Encoding Hold-out test set accuracy: 0.7753

GRU.NN 0.5 Dropout One Hot Encoding ROC AUC 0.844
```



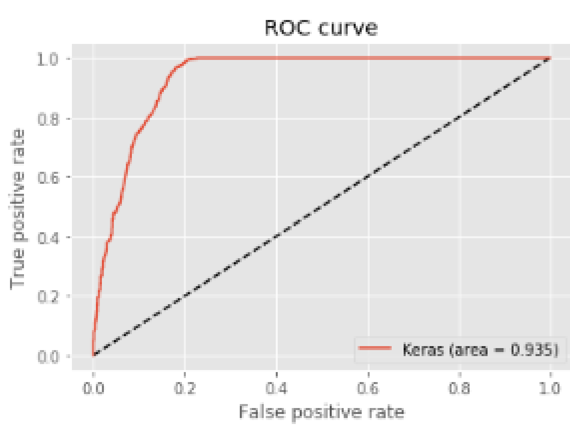




Bidirectional LSTMs

```
res = evaluate_model('Bidirection LSTM One Hot Encoding', bi_lstm_model, max_epochs,
                                    X_train, y_train, X_dev, y_dev,
                                    X_test, y_test, earlystop_callback)
result.append(res)
```

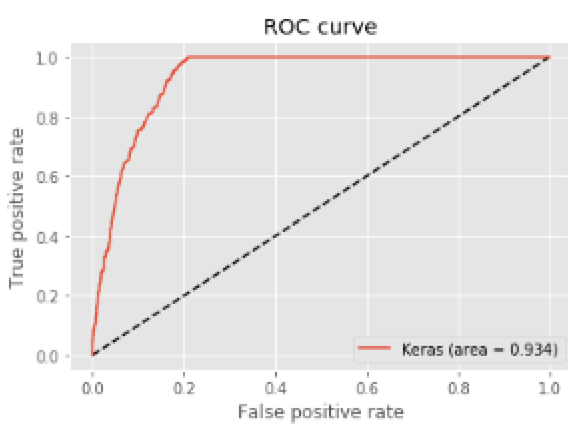
```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 21s - loss: 0.4604 - accuracy: 0.7488 - val_loss: 0.4221 - val_accuracy: 0.7718
Epoch 2/10
13713/13713 - 18s - loss: 0.3791 - accuracy: 0.8135 - val_loss: 0.3764 - val_accuracy: 0.8174
Epoch 3/10
13713/13713 - 18s - loss: 0.3608 - accuracy: 0.8287 - val_loss: 0.3695 - val_accuracy: 0.8236
Epoch 4/10
13713/13713 - 17s - loss: 0.3531 - accuracy: 0.8388 - val_loss: 0.3524 - val_accuracy: 0.8418
Epoch 5/10
13713/13713 - 18s - loss: 0.3303 - accuracy: 0.8590 - val_loss: 0.3276 - val_accuracy: 0.8582
Epoch 6/10
13713/13713 - 18s - loss: 0.2984 - accuracy: 0.8792 - val_loss: 0.2875 - val_accuracy: 0.8841
Epoch 7/10
13713/13713 - 17s - loss: 0.2695 - accuracy: 0.8937 - val_loss: 0.2656 - val_accuracy: 0.8910
Epoch 8/10
13713/13713 - 17s - loss: 0.3149 - accuracy: 0.8554 - val_loss: 0.2709 - val_accuracy: 0.8906
Epoch 9/10
13713/13713 - 18s - loss: 0.2934 - accuracy: 0.8713 - val_loss: 0.3822 - val_accuracy: 0.8159
Epoch 10/10
13713/13713 - 18s - loss: 0.2993 - accuracy: 0.8682 - val_loss: 0.2596 - val_accuracy: 0.8914

Bidirection LSTM One Hot Encoding Time of execution for training (seconds):    179.411

Bidirection LSTM One Hot Encoding Full training set accuracy: 0.9010

Bidirection LSTM One Hot Encoding Development set accuracy: 0.8914

Bidirection LSTM One Hot Encoding Hold-out test set accuracy: 0.8803

Bidirection LSTM One Hot Encoding ROC AUC 0.935
```



Bidirectional LSTMs 0.25

```
res = evaluate_model('Bidirection LSTM 0.25 Dropout One Hot Encoding', bi_lstm_model025, max_epochs,
                                    X_train, y_train, X_dev, y_dev,
                                    X_test, y_test, earlystop_callback)
result.append(res)
```

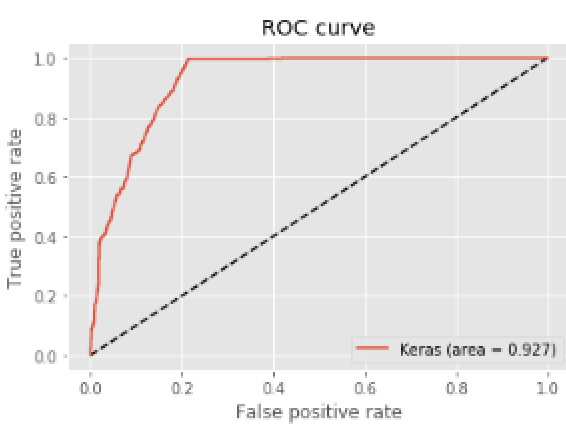
```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 22s - loss: 0.4752 - accuracy: 0.7542 - val_loss: 0.4116 - val_accuracy: 0.7605
Epoch 2/10
13713/13713 - 21s - loss: 0.3887 - accuracy: 0.8121 - val_loss: 0.3859 - val_accuracy: 0.7962
Epoch 3/10
13713/13713 - 23s - loss: 0.3784 - accuracy: 0.8189 - val_loss: 0.3723 - val_accuracy: 0.8112
Epoch 4/10
13713/13713 - 23s - loss: 0.3731 - accuracy: 0.8251 - val_loss: 0.3739 - val_accuracy: 0.8181
Epoch 5/10
13713/13713 - 21s - loss: 0.3666 - accuracy: 0.8294 - val_loss: 0.3594 - val_accuracy: 0.8345
Epoch 6/10
13713/13713 - 20s - loss: 0.3509 - accuracy: 0.8464 - val_loss: 0.3392 - val_accuracy: 0.8480
Epoch 7/10
13713/13713 - 20s - loss: 0.3393 - accuracy: 0.8602 - val_loss: 0.3026 - val_accuracy: 0.8695
Epoch 8/10
13713/13713 - 20s - loss: 0.3279 - accuracy: 0.8617 - val_loss: 0.2854 - val_accuracy: 0.8852
Epoch 9/10
13713/13713 - 19s - loss: 0.2840 - accuracy: 0.8881 - val_loss: 0.2601 - val_accuracy: 0.8914
Epoch 10/10
13713/13713 - 20s - loss: 0.2583 - accuracy: 0.9001 - val_loss: 0.2633 - val_accuracy: 0.8914

Bidirection LSTM 0.25 Dropout One Hot Encoding Time of execution for training (seconds):    209.701

Bidirection LSTM 0.25 Dropout One Hot Encoding Full training set accuracy: 0.9036

Bidirection LSTM 0.25 Dropout One Hot Encoding Development set accuracy: 0.8914

Bidirection LSTM 0.25 Dropout One Hot Encoding Hold-out test set accuracy: 0.8841

Bidirection LSTM 0.25 Dropout One Hot Encoding ROC AUC 0.934
```



Bidirectional LSTMs 0.5

```
res = evaluate_model('Bidirectional LSTM 0.25 Dropout One Hot Encoding', bi_lstm_model05, max_epochs,
                                X_train, y_train, X_dev, y_dev,
                                X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 22s - loss: 0.4755 - accuracy: 0.7364 - val_loss: 0.4073 - val_accuracy: 0.7820
Epoch 2/10
13713/13713 - 19s - loss: 0.4158 - accuracy: 0.7906 - val_loss: 0.3763 - val_accuracy: 0.8082
Epoch 3/10
13713/13713 - 19s - loss: 0.3973 - accuracy: 0.7939 - val_loss: 0.3746 - val_accuracy: 0.8079
Epoch 4/10
13713/13713 - 19s - loss: 0.3999 - accuracy: 0.8018 - val_loss: 0.3740 - val_accuracy: 0.8144
Epoch 5/10
13713/13713 - 19s - loss: 0.3933 - accuracy: 0.8054 - val_loss: 0.3659 - val_accuracy: 0.8225
Epoch 6/10
13713/13713 - 19s - loss: 0.3914 - accuracy: 0.8120 - val_loss: 0.3664 - val_accuracy: 0.8210
Epoch 7/10
13713/13713 - 19s - loss: 0.3876 - accuracy: 0.8109 - val_loss: 0.3499 - val_accuracy: 0.8356
Epoch 8/10
13713/13713 - 19s - loss: 0.3634 - accuracy: 0.8279 - val_loss: 0.3355 - val_accuracy: 0.8498
Epoch 9/10
13713/13713 - 22s - loss: 0.3651 - accuracy: 0.8269 - val_loss: 0.2983 - val_accuracy: 0.8804
Epoch 10/10
13713/13713 - 23s - loss: 0.3100 - accuracy: 0.8601 - val_loss: 0.2702 - val_accuracy: 0.8884

Bidirectional LSTM 0.25 Dropout One Hot Encoding Time of execution for training (seconds):    199.292

Bidirectional LSTM 0.25 Dropout One Hot Encoding Full training set accuracy: 0.8992

Bidirectional LSTM 0.25 Dropout One Hot Encoding Development set accuracy: 0.8884

Bidirectional LSTM 0.25 Dropout One Hot Encoding Hold-out test set accuracy: 0.8797

Bidirectional LSTM 0.25 Dropout One Hot Encoding ROC AUC 0.927
```



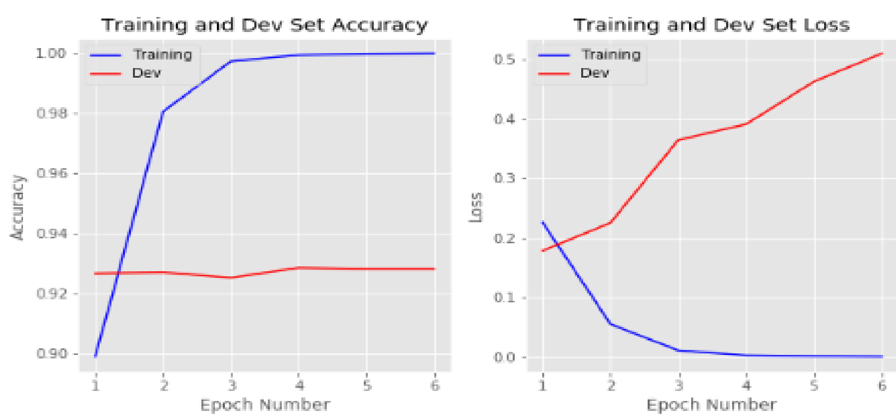
Word Embedding

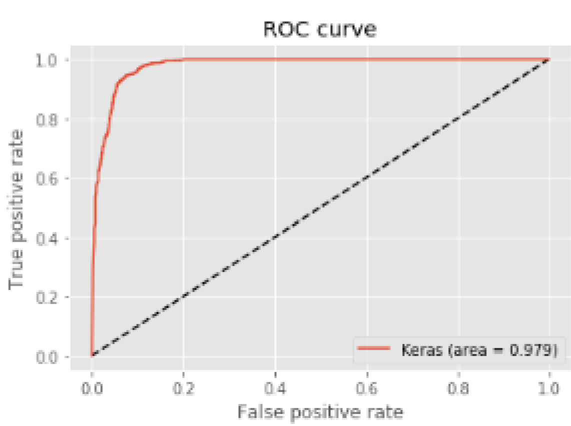
```
train_sequence_we = word_embeddding_encode_docs(tokenizer, max_length, docs = traindocs)
train_sequence_we

#Word embedding
X_train, y_train, X_dev, y_dev, X_test, y_test = train_dev_test_split(train_sequence_we)
```

Conv1D Model Word Embedding

```
res = evaluate_model('Conv1D Word embedding', conv1d_model, max_epochs, X_train, y_train, X_dev, y_dev,
                                                              X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 3s - loss: 0.2256 - accuracy: 0.8990 - val_loss: 0.1782 - val_accuracy: 0.9267
Epoch 2/10
13713/13713 - 3s - loss: 0.0548 - accuracy: 0.9805 - val_loss: 0.2252 - val_accuracy: 0.9271
Epoch 3/10
13713/13713 - 2s - loss: 0.0105 - accuracy: 0.9973 - val_loss: 0.3645 - val_accuracy: 0.9253
Epoch 4/10
13713/13713 - 2s - loss: 0.0023 - accuracy: 0.9994 - val_loss: 0.3910 - val_accuracy: 0.9285
Epoch 5/10
13713/13713 - 2s - loss: 8.0019e-04 - accuracy: 0.9998 - val_loss: 0.4627 - val_accuracy: 0.9282
Epoch 6/10
13713/13713 - 2s - loss: 1.9019e-04 - accuracy: 0.9999 - val_loss: 0.5101 - val_accuracy: 0.9282

Conv1D Word embedding Time of execution for training (seconds):     14.855

Conv1D Word embedding Full training set accuracy: 1.0000

Conv1D Word embedding Development set accuracy: 0.9282

Conv1D Word embedding Hold-out test set accuracy: 0.9273

Conv1D Word embedding ROC AUC 0.979
```

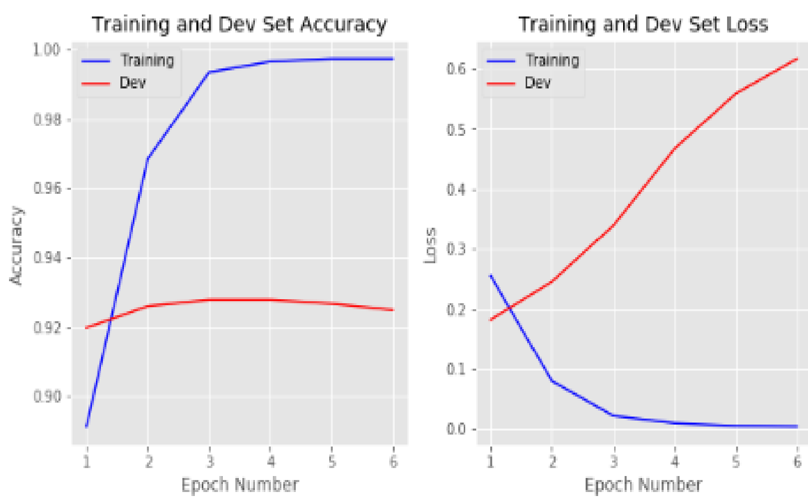


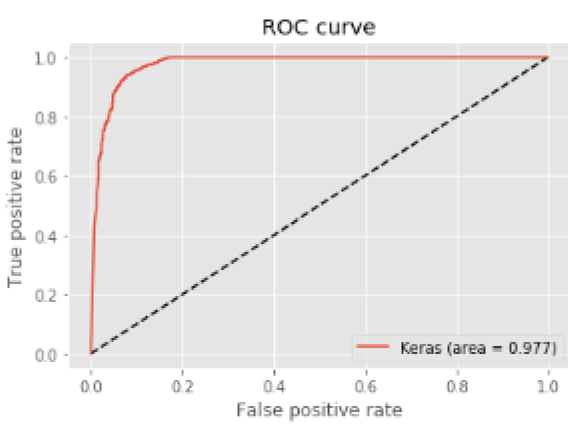


Conv1D Model 0.25 Word Embedding

```
res = evaluate_model('Conv1D 0.25 Dropout Word embedding', conv1d_model025, max_epochs, X_train, y_train, X_dev, y_dev,
                                                         X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 2s - loss: 0.2549 - accuracy: 0.8913 - val_loss: 0.1825 - val_accuracy: 0.9198
Epoch 2/10
13713/13713 - 2s - loss: 0.0799 - accuracy: 0.9684 - val_loss: 0.2453 - val_accuracy: 0.9260
Epoch 3/10
13713/13713 - 2s - loss: 0.0223 - accuracy: 0.9934 - val_loss: 0.3389 - val_accuracy: 0.9278
Epoch 4/10
13713/13713 - 3s - loss: 0.0101 - accuracy: 0.9964 - val_loss: 0.4664 - val_accuracy: 0.9278
Epoch 5/10
13713/13713 - 2s - loss: 0.0054 - accuracy: 0.9972 - val_loss: 0.5587 - val_accuracy: 0.9267
Epoch 6/10
13713/13713 - 2s - loss: 0.0046 - accuracy: 0.9972 - val_loss: 0.6163 - val_accuracy: 0.9249

Conv1D 0.25 Dropout Word embedding Time of execution for training (seconds):     14.622

Conv1D 0.25 Dropout Word embedding Full training set accuracy: 0.9999

Conv1D 0.25 Dropout Word embedding Development set accuracy: 0.9249

Conv1D 0.25 Dropout Word embedding Hold-out test set accuracy: 0.9267

Conv1D 0.25 Dropout Word embedding ROC AUC 0.977
```

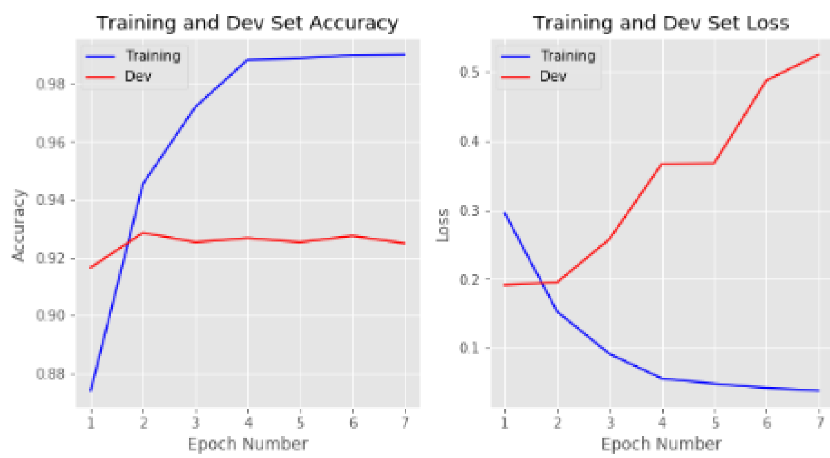


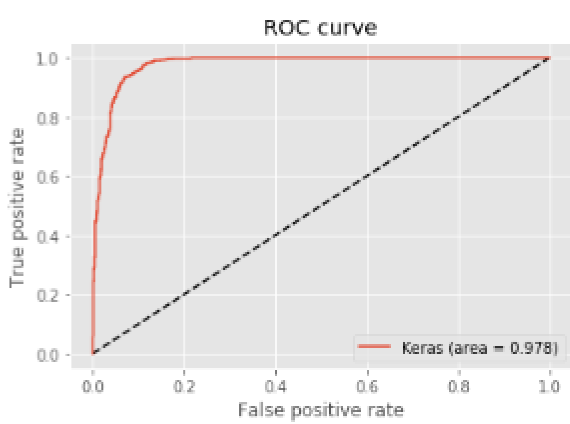


Conv1D Model 0.5 dropout Word Embedding

```
res = evaluate_model('Conv1D 0.5 Dropout Word embedding', conv1d_model05, max_epochs, X_train, y_train, X_dev, y_dev,
                                                      X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 3s - loss: 0.2950 - accuracy: 0.8740 - val_loss: 0.1911 - val_accuracy: 0.9165
Epoch 2/10
13713/13713 - 2s - loss: 0.1526 - accuracy: 0.9455 - val_loss: 0.1949 - val_accuracy: 0.9285
Epoch 3/10
13713/13713 - 3s - loss: 0.0912 - accuracy: 0.9720 - val_loss: 0.2578 - val_accuracy: 0.9253
Epoch 4/10
13713/13713 - 2s - loss: 0.0556 - accuracy: 0.9883 - val_loss: 0.3667 - val_accuracy: 0.9267
Epoch 5/10
13713/13713 - 2s - loss: 0.0481 - accuracy: 0.9888 - val_loss: 0.3674 - val_accuracy: 0.9253
Epoch 6/10
13713/13713 - 2s - loss: 0.0418 - accuracy: 0.9899 - val_loss: 0.4876 - val_accuracy: 0.9275
Epoch 7/10
13713/13713 - 2s - loss: 0.0381 - accuracy: 0.9902 - val_loss: 0.5253 - val_accuracy: 0.9249

Conv1D 0.5 Dropout Word embedding Time of execution for training (seconds):     17.097

Conv1D 0.5 Dropout Word embedding Full training set accuracy: 0.9998

Conv1D 0.5 Dropout Word embedding Development set accuracy: 0.9249

Conv1D 0.5 Dropout Word embedding Hold-out test set accuracy: 0.9317

Conv1D 0.5 Dropout Word embedding ROC AUC 0.978
```

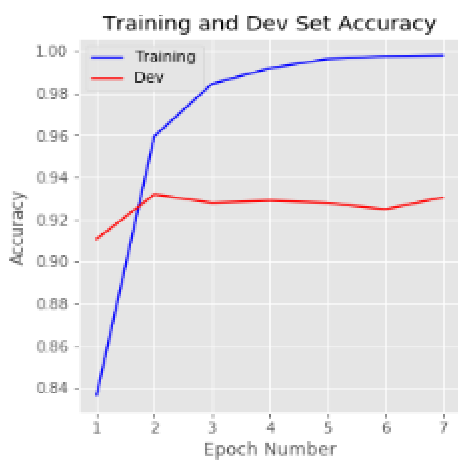


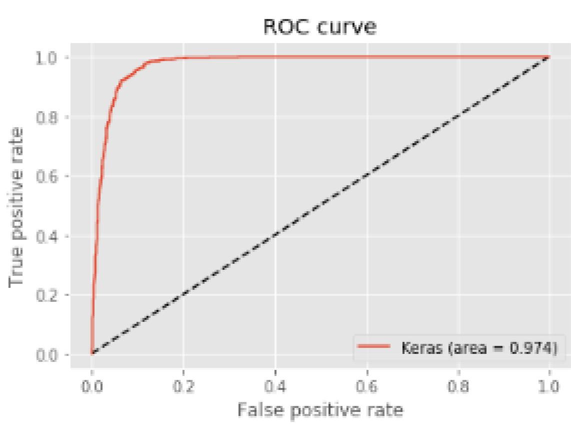


LSTM Model Word Embedding

```
res =evaluate_model('LSTM Word embedding', lstm_model,
                                              max_epochs,
                                              X_train, y_train,
                                              X_dev, y_dev,
                                              X_test, y_test,
                                              earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 15s - loss: 0.3642 - accuracy: 0.8366 - val_loss: 0.2035 - val_accuracy: 0.9107
Epoch 2/10
13713/13713 - 15s - loss: 0.1101 - accuracy: 0.9594 - val_loss: 0.1835 - val_accuracy: 0.9318
Epoch 3/10
13713/13713 - 15s - loss: 0.0498 - accuracy: 0.9844 - val_loss: 0.2366 - val_accuracy: 0.9278
Epoch 4/10
13713/13713 - 15s - loss: 0.0301 - accuracy: 0.9917 - val_loss: 0.2706 - val_accuracy: 0.9289
Epoch 5/10
13713/13713 - 14s - loss: 0.0152 - accuracy: 0.9961 - val_loss: 0.3817 - val_accuracy: 0.9278
Epoch 6/10
13713/13713 - 15s - loss: 0.0097 - accuracy: 0.9974 - val_loss: 0.4889 - val_accuracy: 0.9249
Epoch 7/10
13713/13713 - 15s - loss: 0.0092 - accuracy: 0.9978 - val_loss: 0.4411 - val_accuracy: 0.9304

LSTM Word embedding Time of execution for training (seconds):    104.224

LSTM Word embedding Full training set accuracy: 0.9993

LSTM Word embedding Development set accuracy: 0.9304

LSTM Word embedding Hold-out test set accuracy: 0.9256

LSTM Word embedding ROC AUC 0.974
```

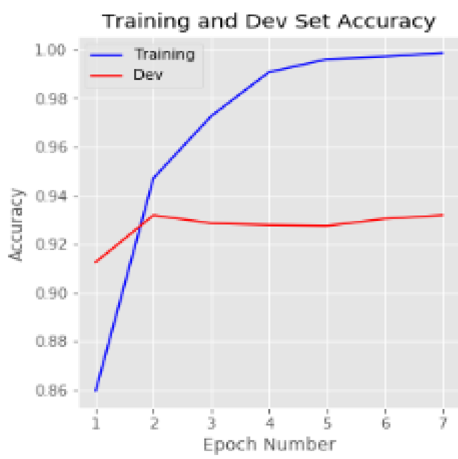


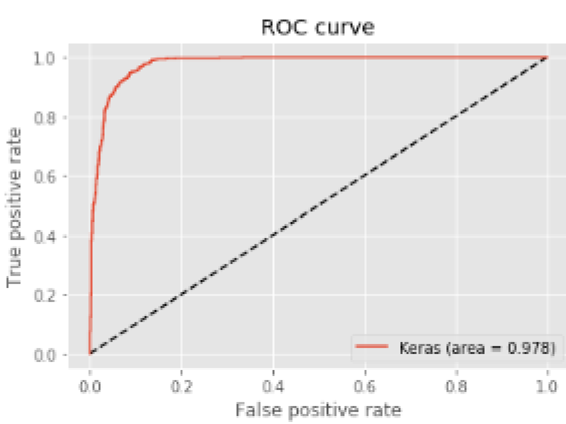


LSTM Model 0.25 dropout Word Embedding

```
res =evaluate_model('LSTM 0.25 Dropout Word embedding', lstm_model025,
                                                max_epochs,
                                                X_train, y_train,
                                                X_dev, y_dev,
                                                X_test, y_test,
                                                earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 15s - loss: 0.3154 - accuracy: 0.8595 - val_loss: 0.2029 - val_accuracy: 0.9125
Epoch 2/10
13713/13713 - 15s - loss: 0.1350 - accuracy: 0.9471 - val_loss: 0.2107 - val_accuracy: 0.9318
Epoch 3/10
13713/13713 - 14s - loss: 0.0720 - accuracy: 0.9727 - val_loss: 0.2705 - val_accuracy: 0.9285
Epoch 4/10
13713/13713 - 13s - loss: 0.0427 - accuracy: 0.9907 - val_loss: 0.2781 - val_accuracy: 0.9278
Epoch 5/10
13713/13713 - 14s - loss: 0.0274 - accuracy: 0.9961 - val_loss: 0.3723 - val_accuracy: 0.9275
Epoch 6/10
13713/13713 - 15s - loss: 0.0221 - accuracy: 0.9972 - val_loss: 0.3088 - val_accuracy: 0.9304
Epoch 7/10
13713/13713 - 14s - loss: 0.0157 - accuracy: 0.9986 - val_loss: 0.4037 - val_accuracy: 0.9318

LSTM 0.25 Dropout Word embedding Time of execution for training (seconds):    101.109

LSTM 0.25 Dropout Word embedding Full training set accuracy: 0.9993

LSTM 0.25 Dropout Word embedding Development set accuracy: 0.9318

LSTM 0.25 Dropout Word embedding Hold-out test set accuracy: 0.9278

LSTM 0.25 Dropout Word embedding ROC AUC 0.978
```

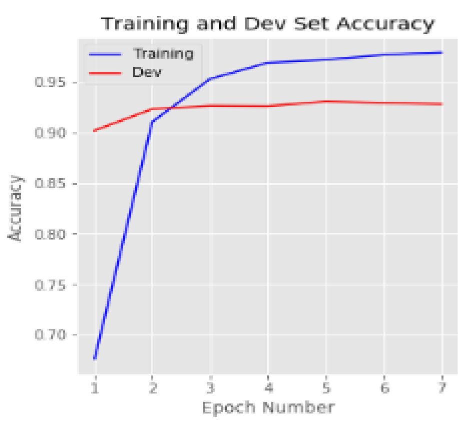


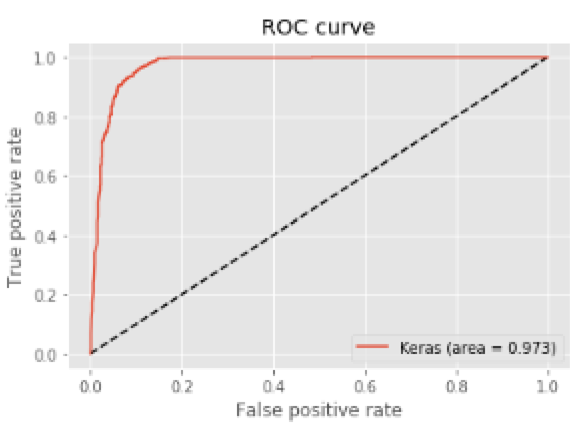


LSTM Model 0.5 dropout Word Embedding

```
res =evaluate_model('LSTM 0.5 Dropout Word embedding', lstm_model05,
                                              max_epochs,
                                              X_train, y_train,
                                              X_dev, y_dev,
                                              X_test, y_test,
                                              earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 13s - loss: 0.5845 - accuracy: 0.6764 - val_loss: 0.2457 - val_accuracy: 0.9019
Epoch 2/10
13713/13713 - 13s - loss: 0.2111 - accuracy: 0.9104 - val_loss: 0.1916 - val_accuracy: 0.9231
Epoch 3/10
13713/13713 - 12s - loss: 0.1164 - accuracy: 0.9528 - val_loss: 0.2445 - val_accuracy: 0.9264
Epoch 4/10
13713/13713 - 12s - loss: 0.0749 - accuracy: 0.9689 - val_loss: 0.2612 - val_accuracy: 0.9260
Epoch 5/10
13713/13713 - 14s - loss: 0.0670 - accuracy: 0.9718 - val_loss: 0.2734 - val_accuracy: 0.9307
Epoch 6/10
13713/13713 - 15s - loss: 0.0477 - accuracy: 0.9766 - val_loss: 0.3798 - val_accuracy: 0.9293
Epoch 7/10
13713/13713 - 13s - loss: 0.0368 - accuracy: 0.9789 - val_loss: 0.5106 - val_accuracy: 0.9282

LSTM 0.5 Dropout Word embedding Time of execution for training (seconds):     92.827

LSTM 0.5 Dropout Word embedding Full training set accuracy: 0.9991

LSTM 0.5 Dropout Word embedding Development set accuracy: 0.9282

LSTM 0.5 Dropout Word embedding Hold-out test set accuracy: 0.9196

LSTM 0.5 Dropout Word embedding ROC AUC 0.973
```

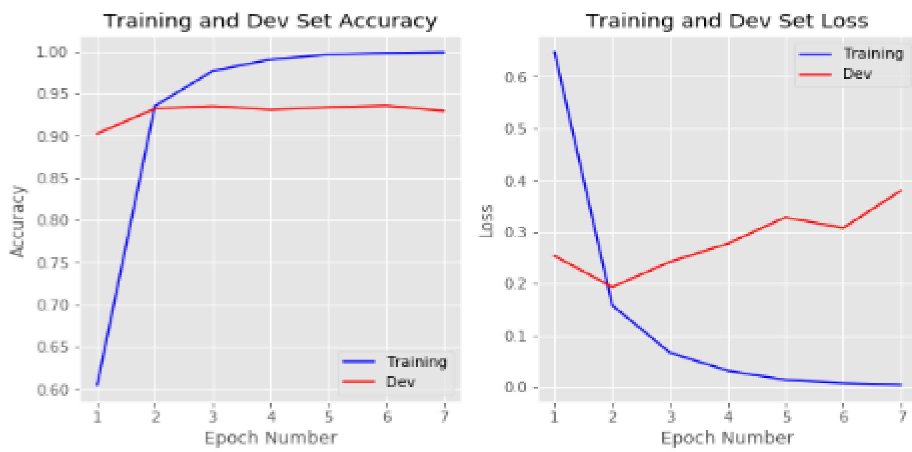


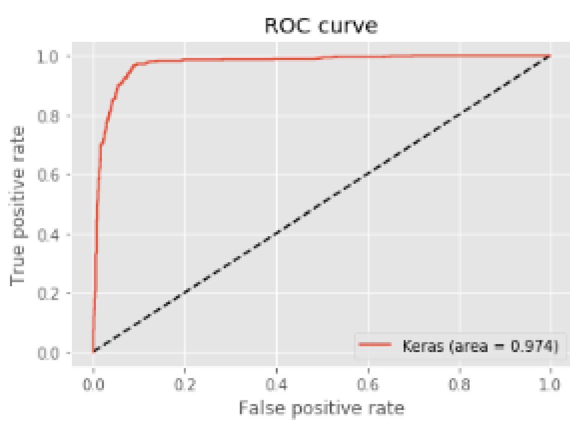


GRU Model Word Embedding

```
res = evaluate_model('GRU.NN Word embedding', gru_model, max_epochs, X_train, y_train,
                     X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 11s - loss: 0.6469 - accuracy: 0.6046 - val_loss: 0.2538 - val_accuracy: 0.9027
Epoch 2/10
13713/13713 - 10s - loss: 0.1584 - accuracy: 0.9354 - val_loss: 0.1935 - val_accuracy: 0.9322
Epoch 3/10
13713/13713 - 10s - loss: 0.0669 - accuracy: 0.9770 - val_loss: 0.2422 - val_accuracy: 0.9351
Epoch 4/10
13713/13713 - 10s - loss: 0.0321 - accuracy: 0.9906 - val_loss: 0.2774 - val_accuracy: 0.9311
Epoch 5/10
13713/13713 - 10s - loss: 0.0144 - accuracy: 0.9964 - val_loss: 0.3284 - val_accuracy: 0.9336
Epoch 6/10
13713/13713 - 10s - loss: 0.0081 - accuracy: 0.9981 - val_loss: 0.3075 - val_accuracy: 0.9358
Epoch 7/10
13713/13713 - 10s - loss: 0.0045 - accuracy: 0.9991 - val_loss: 0.3798 - val_accuracy: 0.9296

GRU.NN Word embedding Time of execution for training (seconds):     71.401

GRU.NN Word embedding Full training set accuracy: 0.9994

GRU.NN Word embedding Development set accuracy: 0.9296

GRU.NN Word embedding Hold-out test set accuracy: 0.9273

GRU.NN Word embedding ROC AUC 0.974
```

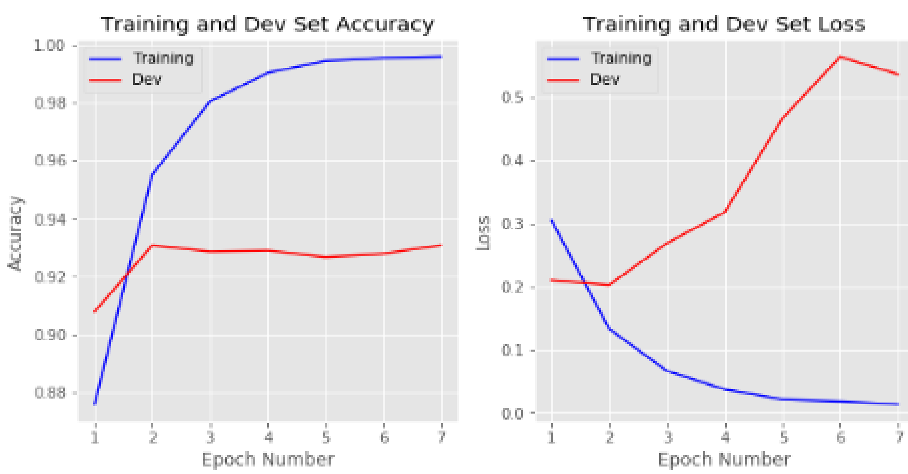


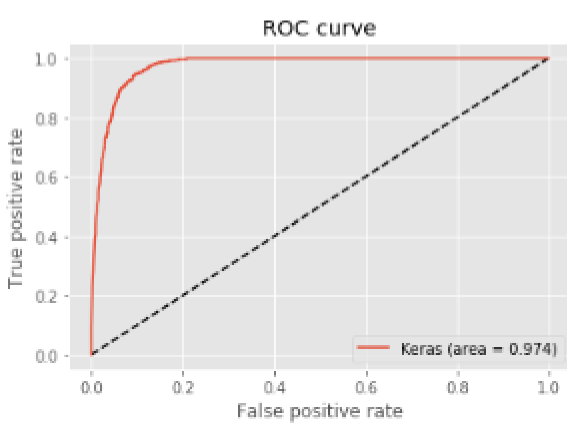


GRU Model 0.25 dropout Word Embedding

```
res = evaluate_model('GRU.NN 0.25 Dropout Word embedding', gru_model025, max_epochs, X_train, y_train,
                     X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 11s - loss: 0.3044 - accuracy: 0.8758 - val_loss: 0.2097 - val_accuracy: 0.9078
Epoch 2/10
13713/13713 - 12s - loss: 0.1324 - accuracy: 0.9552 - val_loss: 0.2023 - val_accuracy: 0.9307
Epoch 3/10
13713/13713 - 12s - loss: 0.0659 - accuracy: 0.9805 - val_loss: 0.2686 - val_accuracy: 0.9285
Epoch 4/10
13713/13713 - 11s - loss: 0.0368 - accuracy: 0.9903 - val_loss: 0.3176 - val_accuracy: 0.9289
Epoch 5/10
13713/13713 - 10s - loss: 0.0208 - accuracy: 0.9945 - val_loss: 0.4663 - val_accuracy: 0.9267
Epoch 6/10
13713/13713 - 10s - loss: 0.0176 - accuracy: 0.9954 - val_loss: 0.5638 - val_accuracy: 0.9278
Epoch 7/10
13713/13713 - 9s - loss: 0.0132 - accuracy: 0.9958 - val_loss: 0.5362 - val_accuracy: 0.9307

GRU.NN 0.25 Dropout Word embedding Time of execution for training (seconds):     75.499

GRU.NN 0.25 Dropout Word embedding Full training set accuracy: 0.9990

GRU.NN 0.25 Dropout Word embedding Development set accuracy: 0.9307

GRU.NN 0.25 Dropout Word embedding Hold-out test set accuracy: 0.9185

GRU.NN 0.25 Dropout Word embedding ROC AUC 0.974
```

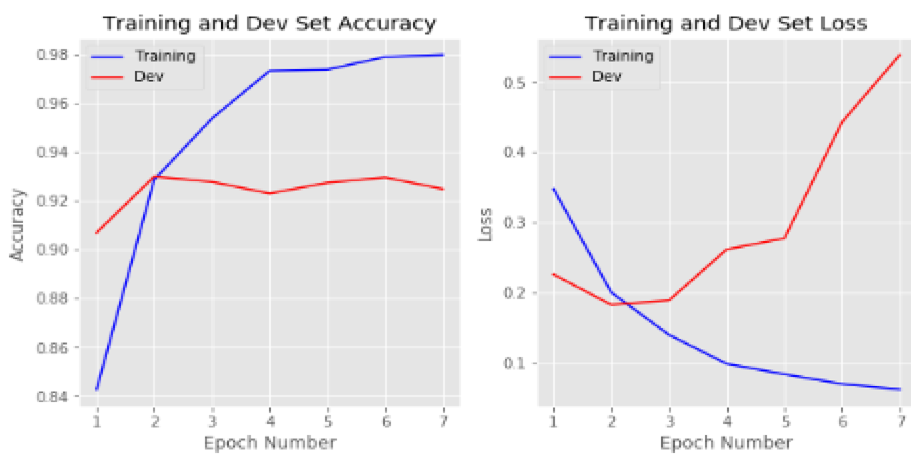


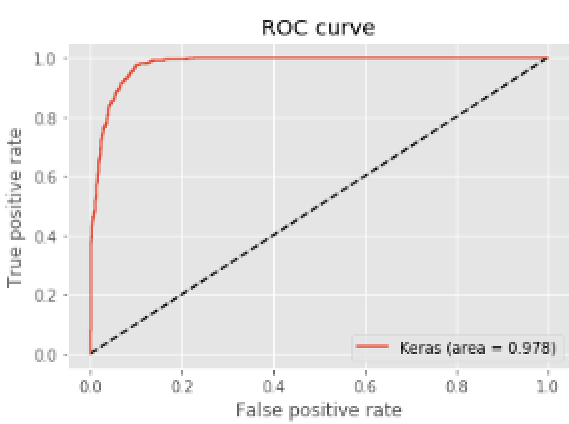


GRU Model 0.5 dropout Word Embedding

```
res = evaluate_model('GRU.NN 0.25 Dropout Word embedding', gru_model05, max_epochs, X_train, y_train,
                           X_dev, y_dev, X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 10s - loss: 0.3472 - accuracy: 0.8427 - val_loss: 0.2256 - val_accuracy: 0.9070
Epoch 2/10
13713/13713 - 10s - loss: 0.1999 - accuracy: 0.9288 - val_loss: 0.1822 - val_accuracy: 0.9300
Epoch 3/10
13713/13713 - 10s - loss: 0.1395 - accuracy: 0.9538 - val_loss: 0.1886 - val_accuracy: 0.9278
Epoch 4/10
13713/13713 - 10s - loss: 0.0982 - accuracy: 0.9734 - val_loss: 0.2613 - val_accuracy: 0.9231
Epoch 5/10
13713/13713 - 10s - loss: 0.0834 - accuracy: 0.9739 - val_loss: 0.2771 - val_accuracy: 0.9275
Epoch 6/10
13713/13713 - 10s - loss: 0.0695 - accuracy: 0.9791 - val_loss: 0.4428 - val_accuracy: 0.9296
Epoch 7/10
13713/13713 - 10s - loss: 0.0616 - accuracy: 0.9799 - val_loss: 0.5386 - val_accuracy: 0.9249

GRU.NN 0.25 Dropout Word embedding Time of execution for training (seconds):     70.299

GRU.NN 0.25 Dropout Word embedding Full training set accuracy: 0.9994

GRU.NN 0.25 Dropout Word embedding Development set accuracy: 0.9249

GRU.NN 0.25 Dropout Word embedding Hold-out test set accuracy: 0.9262

GRU.NN 0.25 Dropout Word embedding ROC AUC 0.978
```

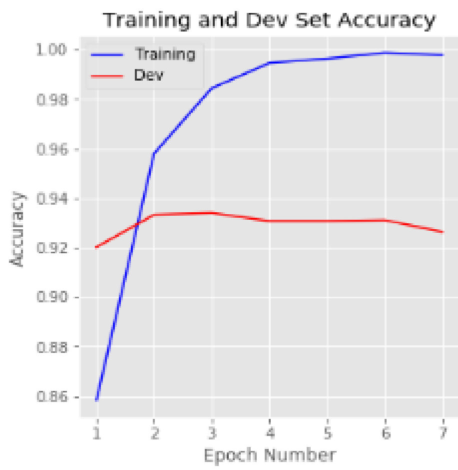


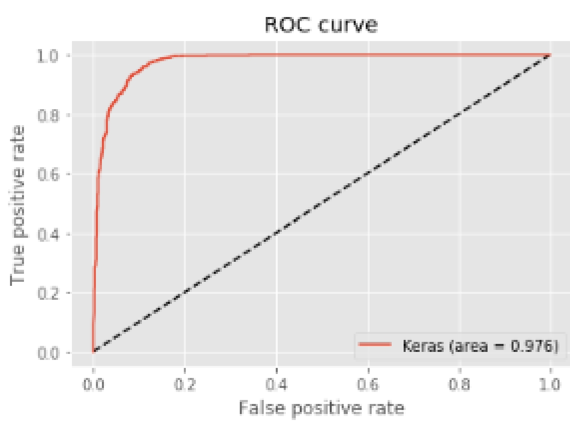


Bidirectional LSTM Word Embedding

```
res = evaluate_model('BidirectionLSTM  Word embedding', bi_lstm_model, max_epochs,
                                  X_train, y_train, X_dev, y_dev,
                                  X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 19s - loss: 0.2957 - accuracy: 0.8584 - val_loss: 0.1840 - val_accuracy: 0.9202
Epoch 2/10
13713/13713 - 17s - loss: 0.1068 - accuracy: 0.9580 - val_loss: 0.1829 - val_accuracy: 0.9333
Epoch 3/10
13713/13713 - 17s - loss: 0.0462 - accuracy: 0.9844 - val_loss: 0.2419 - val_accuracy: 0.9340
Epoch 4/10
13713/13713 - 17s - loss: 0.0192 - accuracy: 0.9946 - val_loss: 0.2593 - val_accuracy: 0.9307
Epoch 5/10
13713/13713 - 17s - loss: 0.0129 - accuracy: 0.9964 - val_loss: 0.3176 - val_accuracy: 0.9307
Epoch 6/10
13713/13713 - 23s - loss: 0.0061 - accuracy: 0.9988 - val_loss: 0.3319 - val_accuracy: 0.9311
Epoch 7/10
13713/13713 - 21s - loss: 0.0075 - accuracy: 0.9979 - val_loss: 0.4375 - val_accuracy: 0.9264

BidirectionLSTM  Word embedding Time of execution for training (seconds):    131.695

BidirectionLSTM  Word embedding Full training set accuracy: 0.9993

BidirectionLSTM  Word embedding Development set accuracy: 0.9264

BidirectionLSTM  Word embedding Hold-out test set accuracy: 0.9207

BidirectionLSTM  Word embedding ROC AUC 0.976
```

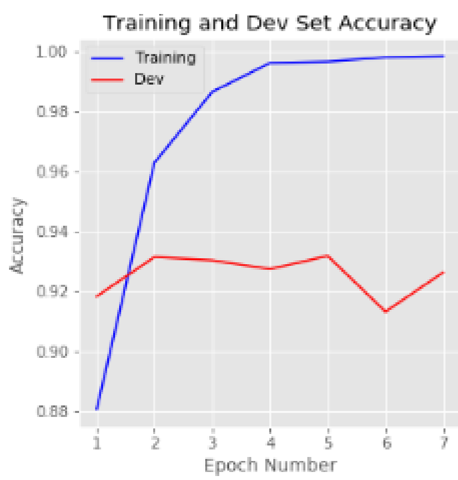


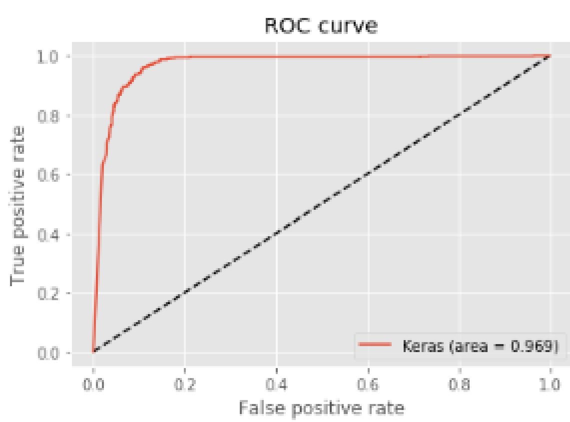


Bidirectional LSTM 0.25 dropout Word Embedding

```
res = evaluate_model('BidirectionLSTM 0.25 dropout Word embedding', bi_lstm_model025, max_epochs,
                                    X_train, y_train, X_dev, y_dev,
                                    X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 19s - loss: 0.2761 - accuracy: 0.8808 - val_loss: 0.1819 - val_accuracy: 0.9183
Epoch 2/10
13713/13713 - 18s - loss: 0.1054 - accuracy: 0.9629 - val_loss: 0.1761 - val_accuracy: 0.9315
Epoch 3/10
13713/13713 - 17s - loss: 0.0456 - accuracy: 0.9865 - val_loss: 0.2675 - val_accuracy: 0.9304
Epoch 4/10
13713/13713 - 18s - loss: 0.0191 - accuracy: 0.9961 - val_loss: 0.4326 - val_accuracy: 0.9275
Epoch 5/10
13713/13713 - 18s - loss: 0.0166 - accuracy: 0.9966 - val_loss: 0.4832 - val_accuracy: 0.9318
Epoch 6/10
13713/13713 - 18s - loss: 0.0123 - accuracy: 0.9980 - val_loss: 0.4446 - val_accuracy: 0.9132
Epoch 7/10
13713/13713 - 17s - loss: 0.0100 - accuracy: 0.9984 - val_loss: 0.7109 - val_accuracy: 0.9264

BidirectionLSTM 0.25 dropout Word embedding Time of execution for training (seconds):    125.597

BidirectionLSTM 0.25 dropout Word embedding Full training set accuracy: 0.9991

BidirectionLSTM 0.25 dropout Word embedding Development set accuracy: 0.9264

BidirectionLSTM 0.25 dropout Word embedding Hold-out test set accuracy: 0.9169

BidirectionLSTM 0.25 dropout Word embedding ROC AUC 0.969
```

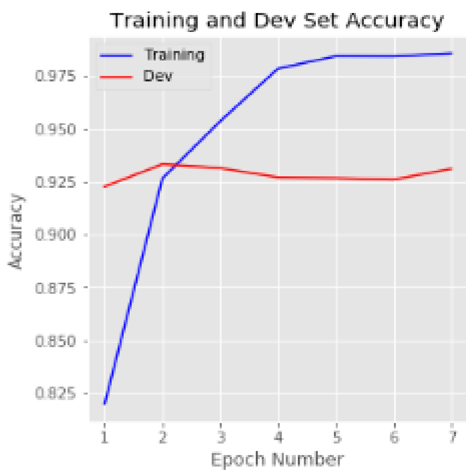


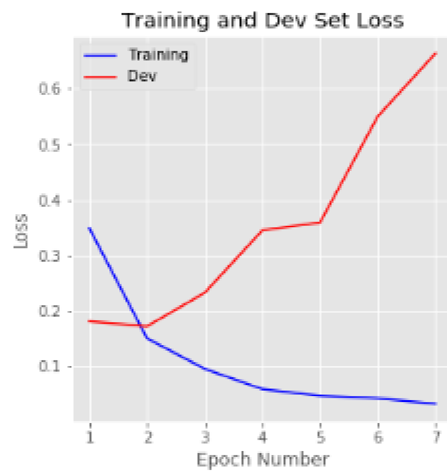


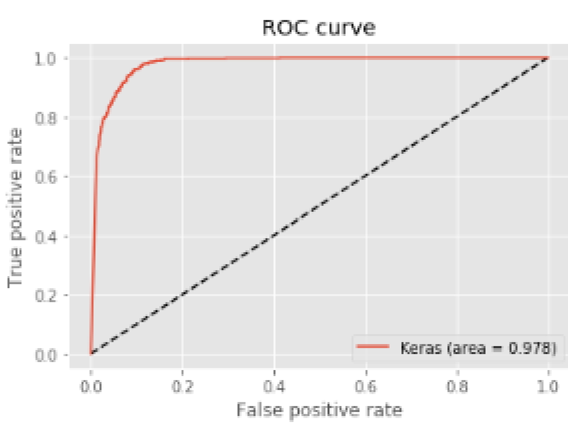
Bidirectional LSTM 0.5 dropout Word Embedding

```
res = evaluate_model('BidirectionLSTM 0.5 dropout Word embedding', bi_lstm_model05, max_epochs,
                              X_train, y_train, X_dev, y_dev,
                              X_test, y_test, earlystop_callback)
result.append(res)
```

```
Train on 13713 samples, validate on 2743 samples
Epoch 1/10
13713/13713 - 18s - loss: 0.3487 - accuracy: 0.8201 - val_loss: 0.1813 - val_accuracy: 0.9227
Epoch 2/10
13713/13713 - 17s - loss: 0.1508 - accuracy: 0.9266 - val_loss: 0.1725 - val_accuracy: 0.9333
Epoch 3/10
13713/13713 - 17s - loss: 0.0959 - accuracy: 0.9535 - val_loss: 0.2329 - val_accuracy: 0.9315
Epoch 4/10
13713/13713 - 17s - loss: 0.0596 - accuracy: 0.9784 - val_loss: 0.3450 - val_accuracy: 0.9271
Epoch 5/10
13713/13713 - 17s - loss: 0.0477 - accuracy: 0.9843 - val_loss: 0.3592 - val_accuracy: 0.9267
Epoch 6/10
13713/13713 - 17s - loss: 0.0431 - accuracy: 0.9842 - val_loss: 0.5497 - val_accuracy: 0.9260
Epoch 7/10
13713/13713 - 17s - loss: 0.0332 - accuracy: 0.9856 - val_loss: 0.6632 - val_accuracy: 0.9311

BidirectionLSTM 0.5 dropout Word embedding Time of execution for training (seconds):    119.248

BidirectionLSTM 0.5 dropout Word embedding Full training set accuracy: 0.9996

BidirectionLSTM 0.5 dropout Word embedding Development set accuracy: 0.9311

BidirectionLSTM 0.5 dropout Word embedding Hold-out test set accuracy: 0.9267

BidirectionLSTM 0.5 dropout Word embedding ROC AUC 0.978
```







Results

```
result_df = pd.DataFrame(result, columns = ['ModelName','Training Execution Time (seconds)','Training Accuracy',\
                                            'Development Accuracy', 'Test Accuracy', 'Area under ROC curve'])
result_df.set_index('ModelName', inplace=True)
result_df =result_df.apply(lambda x:round(x, 4))
result_df.sort_values(['Area under ROC curve', 'Test Accuracy'], ascending=[False, False], inplace=True)
result_df
```

ModelName	Training Execution Time (seconds)	Training Accuracy	Development Accuracy	Test Accuracy	Area under ROC curve
Conv1D Word embedding	14.8550	1.0000	0.9282	0.9273	0.9789
GRU.NN 0.25 Dropout Word embedding	70.2995	0.9994	0.9249	0.9262	0.9785
BidirectionLSTM 0.5 dropout Word embedding	119.2479	0.9996	0.9311	0.9267	0.9783
LSTM 0.25 Dropout Word embedding	101.1095	0.9993	0.9318	0.9278	0.9782
Conv1D 0.5 Dropout Word embedding	17.0970	0.9998	0.9249	0.9317	0.9775
Conv1D 0.25 Dropout Word embedding	14.8220	0.9999	0.9249	0.9267	0.9768
BidirectionLSTM Word embedding	131.6948	0.9993	0.9264	0.9207	0.9764
LSTM Word embedding	104.2240	0.9993	0.9304	0.9256	0.9745
GRU.NN 0.25 Dropout Word embedding	75.4989	0.9990	0.9307	0.9185	0.9742
GRU.NN Word embedding	71.4015	0.9994	0.9296	0.9273	0.9737
LSTM 0.5 Dropout Word embedding	92.8269	0.9991	0.9282	0.9196	0.9727
BidirectionLSTM 0.25 dropout Word embedding	125.5969	0.9991	0.9264	0.9169	0.9693
Bidirection LSTM One Hot Encoding	179.4108	0.9010	0.8914	0.8803	0.9347
Bidirection LSTM 0.25 Dropout One Hot Encoding	209.7008	0.9036	0.8914	0.8841	0.9342
Bidirectional LSTM 0.25 Dropout One Hot Encoding	199.2924	0.8992	0.8884	0.8797	0.9273
LSTM One Hot Encoding	166.9902	0.9033	0.8921	0.8830	0.9204
GRU.NN 0.25 Dropout One Hot Encoding	95.0537	0.8834	0.8720	0.8639	0.9102
LSTM 0.5 Dropout One Hot Encoding	147.0805	0.8901	0.8797	0.8693	0.8893
Conv1D One Hot Encoding	30.2500	0.8065	0.7969	0.8004	0.8865
Conv1D Dropout 0.25 one hot encoding	29.7790	0.8025	0.7944	0.8010	0.8797
Conv1D Dropout 0.25 one hot encoding	28.9740	0.7993	0.7907	0.7862	0.8761
LSTM 0.25 Dropout One Hot Encoding	164.2879	0.7725	0.7594	0.7704	0.8474
GRU.NN 0.5 Dropout One Hot Encoding	94.8350	0.7778	0.7634	0.7753	0.8444
GRU.NN One Hot Encoding	64.2690	0.5743	0.5840	0.5577	0.5806

```
result_df.to_csv('NN_Results.csv')
```