Comparison of Deep Learning Frameworks in Spam Email Filtering  
Xiaodan Li

Department of electrical and computer Engineering

Duke University

1. Introduction

With the development of internet, network security becomes more and more demanding in almost every field. In 2015, IBM and Ponemon Institute conducted research on the cost due to data breach in 62 companies. The average cost of data breach is $6.5 million [1]. Recently, there are many security events such as WannaCry ransomware attack, dark web and so on. The wannacry vulnerability has affected more than 3000,000 computers in 150 countries around the world. There are about 8 billion lost due to WannaCry ransomware attack. Many important organizations such as hospitals, universities and banks are attacked seriously. One of the key step is to infect machines or hosts inside the target network by spam email. In order to prevent the malicious intrusion into organizations or companies, it is really demanding to detect spam email accurately. Some conventional machine learning models are applied such as naïve Bayes classifier, support vector machine. However, the current email filter can not achieve a preferable accuracy which is really vital, since one successful intrusion may lead to security corruption of the whole company, even bankrupt of the company.

In this project, different deep learning frameworks are applied to detect spam email. In the experiment, the performance of those models is compared, including Naïve Bayes classifier, CNN (convolutional neural network), RNN (recurrent neural network), LSTM (Long Short-Term Memory Networks) and CNN-LSTM. Results on CSDMC2010 SPAM corpus [2] indicate that stacked LSTM achieved 99.20% detection accuracy, which is much better than other machine learning algorithms for the problem.

2. Method

2.1 Data preprocessing

As to the dataset, the email is stored by html format. There are many unnecessary contents such as CC, from, to, subject. There are also many html tags like <br>, <p>, <html>, <body> and so on. Those can not provide useful information to identify spam emails. What our model focuses is only the plain text. Therefore, we need to delete some items. Those are shown as followed:

* Delete CC, from, to, subject and only keep body.
* Delete stop words and words usually used in email like email, www, com, http, html, gif and so on.
* Delete the words with length less than 2 or more than 10.
* Delete the words with frequency less than 100.

Then we only parse 100 characters from the modified email. If the length of email is less than 100, we pad shorter email with zeros [3]. The unit RNN/LSTM works on the processed email content instead of sentences in the letter.

2.2 Language model

* Language model for Naïve Bayes Classifier

The Hash maps are used to store the counts of words for good email and spam email separately. The likelihood of words in good and spam emails is computed based on the hash maps.

* Language model for deep learning models

The word embedding is used as model features. There are two steps to transform processed email content into word embedding. The top 5000 most frequent words are selected. The ranking number for the word in Top 5000 words list is used to represent the word when we are training deep learning models. The length of word vector will be 100. The ranking number of the word in Top 5000 list represents the words in the word vector. The entry in word vector follows the word sequence from the modified email content. After we have the word vector, every entry (word) in the word vector will be represented by 32 length vectors. The embedding for every word will be concatenated. Consequently, we will have a matrix with dimension as 100 \* 32 to represent an email.

3. Machine learning models

* Naïve Bayes Classifier

(1)

As shown in equation (1), p(C) is the prior probability (eg. The email is spam or not). is conditional probability of a word given the email spam or benign.

(2)

In the equation (2), Nc is the number of email which is spam or benign. N is the total number of emails.

(3)

In equation (3), count(w, C) is the number of word occurring in email C. count(C) is the number of email C. |V| is the number of word vocabulary [4].

* RNN

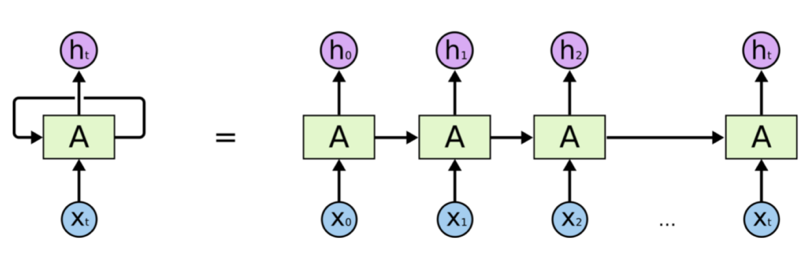


Figure 1. RNN [5]

Figure 1 shows that RNN has a loop. Therefore, its output of previous state will affect current one. The short term memory is allowed to persist due to the structure of RNN [6].

* Basic LSTM cell

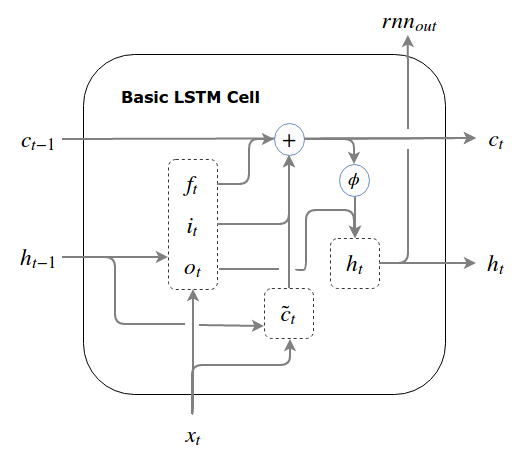


Figure 2. LSTM cell [7]

Figure 2 shows us the internal logic for LSTM cell. It mainly includes three gates: forget gate, input gate and output gate. The forget gate is used to decide which cell state will be thrown away. The input gate decides the states we will update. The output is based on the cell state with some filtering conditions [5].

* Basic LSTM

Dense Layer

LSTM

LSTM

Word Embedding

Output Layer

Figure 3. Structure of basic LSTM

In figure 3, the word Embedding from language model is used to describe email content. Then the word representation will be feed into LSTM. The dense layer with a single neuron and a sigmoid activation function is applied to features processed from LSTM to make 0 or 1 predictions, since this is binary classification problem. The output layer will give us the binary result (spam email or not).

* LSTM with dropout

Dense Layer

LSTM

Word Embedding

Dropout

Dropout

Output Layer

Figure 4. Structure of LSTM with dropout

In order to reduce model complexity and avoid over-fitting, dropout layers are added to the current basic LSTM model.

* CNN

Dense Layer

Conv1D

Conv1D

MaxPooling1D

Word Embedding

Output Layer

Figure 5. Structure of CNN

Figure 5 shows the structure of CNN. The 1D convolutional layer is applied to process features.

* CNN-LSTM

Dense Layer

MaxPooling1D

Word Embedding

Con1D

LSTM

Output Layer

Figure 6. Structure of CNN-LSTM

In figure 6, CNN-LSTM is applied to spam email filtering. Firstly, CNN is used to extract features from word embedding. Then LSTM classifies the email according to the features from CNN [6].

* Stacked LSTM

Dense Layer

LSTM

Word Embedding

LSTM

LSTM

Output Layer

Figure 7. Structure of stacked LSTM

As shown in figure 7, three layers of LSTMs are stacked together to process word embedding. Due to the complexity of this model, it can extract more valuable features.

4. Results and analysis

4.1 Dataset

The dataset is CSDMC2010. This dataset has collected spam email and benign email. In the dataset, there are 4327 messages which include 2949 non-spam messages (HAM) and 1378 spam messages (SPAM) [2]. The sample of spam and non-spam emails are shown in the table. Table 1 shows the example of spam email and non-spam email. The non-spam email is about some normal issues, while spam email usually includes some illegal content.

Table 1. Email examples

|  |  |  |
| --- | --- | --- |
| Email Category | Spam | Non-spam |
| Email Content | <font color=3D"#FFFF00">  This WEEK: Sydney <B>Bares ALL</b> in the park!<br><br>  <font color=3D"#FFFFFF">  Join her in our Live Teen Chat!<br><br>  <font color=3D"#FFFF00">  Watch as Sandy <b>Strips Naked </b>in her <b>Dorm</b>!<br><br>  <font color=3D"#FFFFFF">  Best of All, see it…… | <body>  This is a reminder, sent out once a month, about your linux.ie mailing list memberships. It includes your subscription info and how to use it to change it or unsubscribe from a list.  You can visit the URLs to change your membership status or configuration, including unsubscribing, setting digest-style delivery or disabling delivery altogether (e.g., for a vacation), and so on…… |

In this dataset, the hard label is adopted, since there are only two email categories. In order to compare the performance of machine learning models, dataset is divided into 80% training set and 20% testing set. The training set is used to train the models and get the optimized hyper-parameters. Then we can get the detection accuracy and roc curves for every model by applying our models to testing data. The the settings of hyper-parameters are shown as followed:

embedding\_vector\_length = 32

numHidden = 100

epochs = 40

batch\_size = 64

dropoutRate = 0.2

timesteps = 10

min\_word\_freq = 100

max\_review\_length = 100

4.2 Detection accuracy and analysis

RNN, basic LSTM, LSTM with dropout, CNN, CNN-LSTM, stacked LSTM and Naïve Bayes classifier are applied to CSDMC2010. Our baseline model is Naïve Bayes classifier. The detection accuracy is shown in table 2. In table 2, we can find out that generally, deep learning models perform better than Naïve Bayes. The detection accuracy of stacked LSTM is 11.28% better than Naïve Bayes.

Table 2. Detection Accuracy with word vector length as 100

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Naïve Bayes** | **RNN** | **CNN** | **Basic LSTM** | **LSTM with dropout** | **CNN**  **LSTM** | **Stacked**  **LSTM** |
| **Accuracy** | 87.92% | 88.50% | 98.66% | 97.73% | 98.66% | 98.40% | 99.20% |

As to Naïve Bayes classifier, based on the equations (1) (2) (3), we can find out that this model only considers the frequency of word occurrence in spam or benign emails instead of some long term correlation among different sentences in email. In addition, it fails to collect enough features as what CNN or LSTM does. Therefore, this model misses much useful information. Another difference is language model. In terms of Naïve Bayes classifier, we only record the occurrence of words in emails. However, as to the language model for deep learning model, although the length of word vector is fixed, the sequence of words is kept in word embedding. Therefore, its performance is worse than stacked LSTM, CNN-LSTM.

As to CNN, above the word embedding layer, two 1D convolution layers are set with the number output of filters as 64, kernel size as 3, activation function as rectified linear unit (Relu). Then the max pooling operation is applied to the output from those two 1D convolution layers. The result of max pooling is processed by another two 1D convolution layers with number of filters as 128, kernel size as 3, activation function as rectified linear unit (Relu). The global average pooling is operated on this output. Our problem is binary classification. Therefore, dense layer with one neuron and a sigmoid activation function is built on the result from global average pooling. In order to avoid model over-fitting, the dropout layer is set between dense layer and global average pooling. Due to its structure, CNN can extract features from word embedding efficiently and accurately. CNN keeps feature maps in the beginning. Then it decreases the number of features sharply by using pooling and convolution, which can avoid learning some unrelated noise [6]. Due to its high performance in learning relevant features and ruling out irrelevant features, CNN is adopted as feature extractor and its output is feed into LSTM. The combination of CNN and LSTM has a remarkable detection accuracy which is close to CNN.

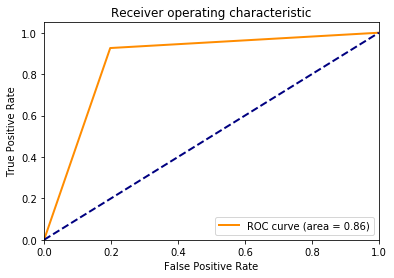
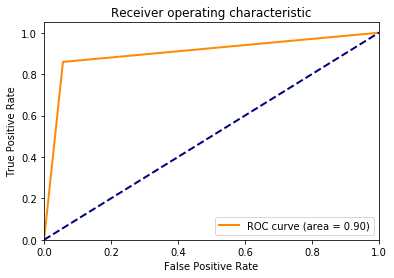
As to the basic LSTM, we only have three layers: word embedding, LSTM, dense layer and output layer. The length of word embedding is 32. In the LSTM layer, 100 memory units are used. The dense layer is then added on the LSTM since this problem is binary classification. Sigmoid function is applied to predict whether the email is spam or not. Then we also add dropout layers between word embedding layer, dense layer and LSTM separately to explore performance change. According to the result, we can find out the detection accuracy increases by about 1% after applying dropout. After we apply dropout to basic LSTM, the model over-fitting is avoided which betters detection performance. We also tried stacked LSTM model. Based on the basic LSTM, two LSTM layers are added. The detection accuracy of stacked LSTM is almost 100%. This is because mul-layer LSTM can extract more important features from the word embedding.

In the table 1, we can also find out RNN performs slightly better than Naïve Bayes. Although RNN can incorporate previous information, it fails to take advantage of long term information [5]. However, in the email, most of sentences are related. Therefore, RNN does not have visible improvement compared to our baseline model.

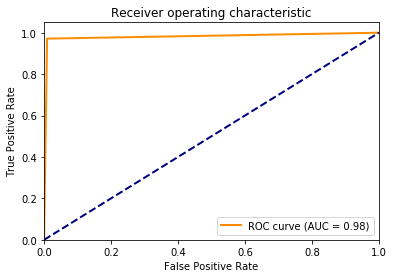
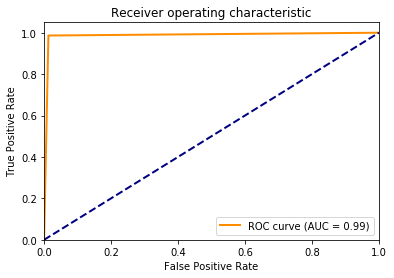
4.3 Roc curves and analysis

The roc curves for each model are shown in Figure 8. Generally, deep learning models perform better than Naïve Bayes. The AUC for LSTM-CNN, stacked LSTM and CNN are 0.99. This means that CNN-LSTM, CNN and stacked LSTM dominate in term of false positive rate and true positive rate. All of them have a low false positive rate in the meantime high true positive rate.

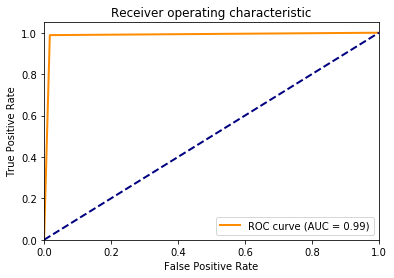
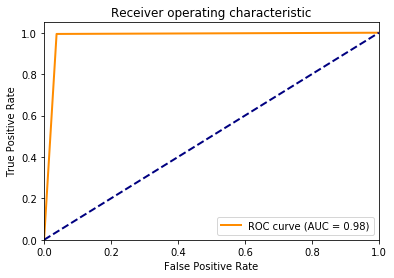
The AUCs of CNN, CNN-LSTM and stacked LSTM are almost 1.0. We also can find out that at the very beginning, those models have a high true positive rate. Then it tends to go flat with the increase of false positive rate. Those models can extract enough features automatically which contribute to high detection performance. As to Naïve Bayes classifier, at first, true positive rate is not good. However, with the increase of false positive rate, it reaches to a high value. The AUC is 0.90 which means the performance of Naïve Bayes is good. In terms of RNN, at the start, it takes a long process for true positive rate to achieve a high value. Therefore, its AUC is even less than baseline model. RNN does not take into consideration the long term dependence among different words or sentences in emails.



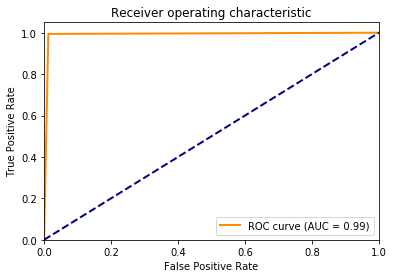
(a) Naïve Bayes (b) RNN



(c) CNN (d) Basic LSTM

(e) LSTM with dropout (f) CNN-LSTM



(g) Stacked LSTM

Figure 8. ROC curves

4.4 Comparison of detection performance with different word vector length

In order to compare the detection performance with different word vector length, we also conduct experiment when word length is 500. The detection accuracy is shown as followed in table 3. We can find that the length of word vector length affects the performance of LSTMs. The length of processed email content is usually less than 500. When we pad the word vector, we add many zeros into the word vector which is too sparse. The performance of basic LSTM is even worse than Naïve Bayes by 17.06%. When we apply the dropout to basic LSTM, it is better. This is because dropout just throw away some connections among different neurons corresponding to padding zeros from input. The stacked LSTM fails to have a proper classification. However, CNN, CNN-LSTM work well under this condition. Therefore, CNN is good at dealing with sparse input vector.

Table 3. Detection Accuracy with word vector length as 500

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Naïve Bayes** | **RNN** | **CNN** | **Basic LSTM** | **LSTM with dropout** | **CNN**  **LSTM** | **Stacked**  **LSTM** |
| **Accuracy** | 87.92% | 88.50% | 98.80% | 70.86% | 87.30% | 98.26% | -- |

5. Discussion and Conclusion

5.1 Conclusion

In this project, different deep learning frameworks are applied to detect spam email. In the experiment, the performance of those models is compared, including Naïve Bayes classifier, CNN (convolutional neural network), RNN (recurrent neural network), LSTM (Long Short-Term Memory Networks), CNN-LSTM and stacked LSTM. There are some interesting observations:

* In terms of language model, though our word embedding needs fixed vector length, it keeps the sequence of words effectively which contributes to the high detection performance of deep learning models such as CNN, CNN-LSTM and stacked LSTM.
* Stacked LSTM is the most preferable spam email filtering model which improve the detection accuracy by 11.28 % with lower false positive and false negative rates. Due to model complexity, stacked LSTM can extract important features perfectly. What is more, CNN-LSTM also performs well. By using pooling and convolution, CNN decreases the number of features sharply. CNN can extract relevant patterns and rule out irrelevant patterns from word embedding automatically and efficiently.
* RNN has slight improvement compared to baseline model. RNN can process the short term memory, but it can not keep long term dependence among different words or sentences. However, words or sentences are highly related in the email. Therefore, it is no better than Naïve Bayes classifier.
* The length of word vector matters in terms of model performance. If there are so many zeros in the input, it will affect the performance of LSTM related model. However, CNN related model can fix this problem properly.
  1. Discussion of applying RNN/LSTM with various word vector length and without cropping

This project is mainly based on keras. In order to make implementation more convenient and simpler, keras accepts the sequence with same length in a batch when RNN/LSTM is adopted. If the length of sequence is not the same, we usually pad the sequence as what I have done in the project. The second reason to use fixed length is that usually the email is short. Therefore, we can just set up a fixed input length. The reason to crop the email is that the email is stored by html format. There are many unnecessary contents such as CC, from, to, subject. There are also many html tags like <br>, <p>, <html>, <body> and so on. Those can not provide useful information to identify spam emails. What our model focuses is only the plain text.

* The potential advantage of RNN/LSTM without cropping emails

However, if we can feed RNN/LSTM with various length and less modification, we can feed the information into models as complete as possible. RNN/LSTM gets enough information. This model can catch the dependence among different sentences, while the model in the report can only capture relation among different emails. This model has finer granulation. Therefore, it can perform better than the current model, even though current detection accuracy is close to 100%.

* How to apply RNN/LSTM without cropping emails

Instead of preprocessing the email, we just get the email content from email body and do not make any modification which then will be feed into RNN/LSTM models. In order to feed model with various length, firstly we need to set up a unit length. The sentences in email will be divided into batches with unit length before being used in keras. The following steps will be the same as what is described in this report. The difference is that the unit RNN/LSTM runs on is sentences instead of content from a whole email. However, this way might break the sentence logic which might affect classification performance. Another way is to add masking layer to the input. This method is usually applied to sequence to sequence temporal sequence problem. If all of the values in the input at certain timestamp are mask value, this sequence will be masked.

Generally, stacked LSTM is the most preferable spam email filter. As to the future work, the training and testing time will be studied. The email filter is real-time system. If the filtering process takes so much time, it will affect email user experience. After that, we will apply the current models to malware detection.

Reference

* [1] IBM. “2015 Cost of Data Breach Study: United States [Internet].” 2015.
* [2] http://www.csmining.org/index.php/spam-email-datasets-.html.
* [3] <https://machinelearningmastery.com/cnn-long-short-term-memory-networks/>.
* [4] Dan Jurafsky and James H Martin. Speech and language processing, 2017. URL https: //web.stanford.edu/~jurafsky/slp3.
* [5] http://colah.github.io/posts/2015-08-Understanding-LSTMs/.
* [6] <https://blog.statsbot.co/text-classifier-algorithms-in-machine-learning-acc115293278>.
* [7]https://www.bing.com/images/search?view=detailV2&ccid=RSTX%2bgrP&id=422D6AF4B88FB8AB6D430166A29E86B8C403A75C&thid=OIP.RSTX-grPK55-Zk4-9j\_1ewHaGo&mediaurl=https%3a%2f%2fr2rt.com%2fstatic%2fimages%2fNH\_BasicLSTMCell.png&exph=471&expw=526&q=lstm+cell&simid=608003539568951832&selectedIndex=0&ajaxhist=0.

Source code:

The source code for RNN filter, LSTMs filter, Naïve Bayes filter are provided as followed:

RNN\_email\_filtering.py

import email

import os

import re

import numpy as np

import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow.python.framework import ops

import NLP\_module

import html2text

from nltk.corpus import stopwords

ops.reset\_default\_graph()

"""

https://github.com/nfmcclure/tensorflow\_cookbook/blob/master/09\_Recurrent\_Neural\_Networks/02\_Implementing\_RNN\_for\_Spam\_Prediction/02\_implementing\_rnn.py

"""

# Create a text cleaning function

def clean\_text(text\_string):

h = html2text.HTML2Text()

h.ignore\_links = True

h.escape\_snob = True

txt = h.handle(text\_string)

txt = txt.lower()

p = re.compile('\W+')

splits = p.split(txt)

result = ""

start\_to\_parse = False

for word in splits:

if NLP\_module.strCmp(word, "Date"):

start\_to\_parse = True

if start\_to\_parse == False:

continue

found = re.search('[0-9]+', word)

if found != None:

continue

stop\_wds = stopwords.words("english")

stop\_wds.extend(["email", "www", "com", "http", "html", "gif", "smtp", "sender", "received", "zzzz", "yyyy","localhost", "org", "esmtp", "debian", "return", "path"])

if word in stop\_wds:

continue

if len(word) <= 2 or len(word) >= 10:

continue

result = result + " " + word

text\_string = result.lower()

return(text\_string)

# Start a graph

sess = tf.Session()

# Set RNN parameters

epochs = 50

batch\_size = 200

max\_sequence\_length = 100

rnn\_size = 10

embedding\_size = 50

min\_word\_frequency = 10

learning\_rate = 0.0005

dropout\_keep\_prob = tf.placeholder(tf.float32)

labels = dict()

label\_path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/SPAMTrain.label'

infile = open(label\_path,'r')

label\_List = list()

for line in infile:

tp = line.split(" ")[1]

eml\_name = tp.split("\n")[0]

labels[eml\_name] = line.split(" ")[0]

label\_List.append(line.split(" ")[0])

infile.close()

path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/training\_new'

listing = os.listdir(path)

listing = listing

fail\_IO = list()

gd\_cnt = 0

bad\_cnt = 0

text\_target = list()

text\_data\_train = list()

for i in range(len(listing)):

fle = listing[i]

if str.lower(fle[-3:])=="eml":

try:

msg = email.message\_from\_file(open(path + '/' + fle))

strs = msg.as\_string()

cleantext = clean\_text(strs)

text\_data\_train.append(cleantext)

if labels[fle] == "1":

gd\_cnt = gd\_cnt + 1

text\_target.append(1)

else:

bad\_cnt = bad\_cnt + 1

text\_target.append(0)

except UnicodeDecodeError:

fail\_IO.append(fle)

continue

# Clean texts

#text\_data\_train = [clean\_text(x) for x in text\_data\_train]

# Change texts into numeric vectors

vocab\_processor = tf.contrib.learn.preprocessing.VocabularyProcessor(max\_sequence\_length,

min\_frequency=min\_word\_frequency)

text\_processed = np.array(list(vocab\_processor.fit\_transform(text\_data\_train)))

# Shuffle and split data

text\_processed = np.array(text\_processed)

text\_data\_target = np.array(text\_target)

#text\_data\_target = np.array([1 if x=='ham' else 0 for x in text\_data\_target])

shuffled\_ix = np.random.permutation(np.arange(len(text\_data\_target)))

x\_shuffled = text\_processed[shuffled\_ix]

y\_shuffled = text\_data\_target[shuffled\_ix]

# Split train/test set

ix\_cutoff = int(len(y\_shuffled)\*0.80)

x\_train, x\_test = x\_shuffled[:ix\_cutoff], x\_shuffled[ix\_cutoff:]

y\_train, y\_test = y\_shuffled[:ix\_cutoff], y\_shuffled[ix\_cutoff:]

vocab\_size = len(vocab\_processor.vocabulary\_)

print("Vocabulary Size: {:d}".format(vocab\_size))

print("80-20 Train Test split: {:d} -- {:d}".format(len(y\_train), len(y\_test)))

# Create placeholders

x\_data = tf.placeholder(tf.int32, [None, max\_sequence\_length])

y\_output = tf.placeholder(tf.int32, [None])

# Create embedding

embedding\_mat = tf.Variable(tf.random\_uniform([vocab\_size, embedding\_size], -1.0, 1.0))

embedding\_output = tf.nn.embedding\_lookup(embedding\_mat, x\_data)

#embedding\_output\_expanded = tf.expand\_dims(embedding\_output, -1)

# Define the RNN cell

#tensorflow change >= 1.0, rnn is put into tensorflow.contrib directory. Prior version not test.

if tf.\_\_version\_\_[0]>='1':

cell=tf.contrib.rnn.BasicRNNCell(num\_units = rnn\_size)

else:

cell = tf.nn.rnn\_cell.BasicRNNCell(num\_units = rnn\_size)

output, state = tf.nn.dynamic\_rnn(cell, embedding\_output, dtype=tf.float32)

output = tf.nn.dropout(output, dropout\_keep\_prob)

# Get output of RNN sequence

output = tf.transpose(output, [1, 0, 2])

last = tf.gather(output, int(output.get\_shape()[0]) - 1)

weight = tf.Variable(tf.truncated\_normal([rnn\_size, 2], stddev=0.1))

bias = tf.Variable(tf.constant(0.1, shape=[2]))

logits\_out = tf.matmul(last, weight) + bias

# Loss function

losses = tf.nn.sparse\_softmax\_cross\_entropy\_with\_logits(logits=logits\_out, labels=y\_output) # logits=float32, labels=int32

loss = tf.reduce\_mean(losses)

accuracy = tf.reduce\_mean(tf.cast(tf.equal(tf.argmax(logits\_out, 1), tf.cast(y\_output, tf.int64)), tf.float32))

# prediction result

pred\_correction = tf.equal(tf.argmax(logits\_out, 1), tf.cast(y\_output, tf.int64))

prediction = tf.argmax(logits\_out, 1)

optimizer = tf.train.RMSPropOptimizer(learning\_rate)

train\_step = optimizer.minimize(loss)

init = tf.global\_variables\_initializer()

sess.run(init)

train\_loss = []

test\_loss = []

train\_accuracy = []

test\_accuracy = []

test\_pred = list()

test\_predcor = list()

# Start training

for epoch in range(epochs):

# Shuffle training data

shuffled\_ix = np.random.permutation(np.arange(len(x\_train)))

x\_train = x\_train[shuffled\_ix]

y\_train = y\_train[shuffled\_ix]

num\_batches = int(len(x\_train)/batch\_size) + 1

# TO DO CALCULATE GENERATIONS ExACTLY

for i in range(num\_batches):

# Select train data

min\_ix = i \* batch\_size

max\_ix = np.min([len(x\_train), ((i+1) \* batch\_size)])

x\_train\_batch = x\_train[min\_ix:max\_ix]

y\_train\_batch = y\_train[min\_ix:max\_ix]

# Run train step

train\_dict = {x\_data: x\_train\_batch, y\_output: y\_train\_batch, dropout\_keep\_prob:0.5}

sess.run(train\_step, feed\_dict=train\_dict)

# Run loss and accuracy for training

temp\_train\_loss, temp\_train\_acc = sess.run([loss, accuracy], feed\_dict=train\_dict)

train\_loss.append(temp\_train\_loss)

train\_accuracy.append(temp\_train\_acc)

# Run Eval Step

test\_dict = {x\_data: x\_test, y\_output: y\_test, dropout\_keep\_prob:1.0}

temp\_test\_loss, temp\_test\_acc, temp\_test\_predcor, temp\_test\_prediction = sess.run([loss, accuracy, pred\_correction, prediction], feed\_dict=test\_dict)

test\_loss.append(temp\_test\_loss)

test\_accuracy.append(temp\_test\_acc)

test\_predcor.append(temp\_test\_predcor)

test\_pred.append(temp\_test\_prediction)

print('Epoch: {}, Test Loss: {:.2}, Test Acc: {:.2}'.format(epoch+1, temp\_test\_loss, temp\_test\_acc))

# Plot loss over time

epoch\_seq = np.arange(1, epochs+1)

plt.plot(epoch\_seq, train\_loss, 'k--', label='Train Set')

plt.plot(epoch\_seq, test\_loss, 'r-', label='Test Set')

plt.title('Softmax Loss')

plt.xlabel('Epochs')

plt.ylabel('Softmax Loss')

plt.legend(loc='upper left')

plt.show()

# Plot accuracy over time

plt.plot(epoch\_seq, train\_accuracy, 'k--', label='Train Set')

plt.plot(epoch\_seq, test\_accuracy, 'r-', label='Test Set')

plt.title('Test Accuracy')

plt.xlabel('Epochs')

plt.ylabel('Accuracy')

plt.legend(loc='upper left')

plt.show()

# get the RoC curve

predcor = test\_predcor[-1]

#print(predcor)

prediction = test\_pred[-1]

#print(prediction)

NLP\_module.plotRoc(prediction, y\_test)

##1 stands for a HAM and 0 stands for a SPAM

detec\_spam = 0

non\_detec\_spam = 0

detec\_good = 0

non\_detec\_good = 0

for i in range(len(prediction)):

if predcor[i] == True:

if y\_test[i] == 0:

detec\_spam = detec\_spam + 1

else:

detec\_good = detec\_good + 1

if predcor[i] == False:

if y\_test[i] == 1:

non\_detec\_good = non\_detec\_good + 1

else:

non\_detec\_spam = non\_detec\_spam + 1

total\_spam = detec\_spam + non\_detec\_spam

total\_good = detec\_good + non\_detec\_good

spam\_rate = 1.0 \* detec\_spam / (detec\_spam + non\_detec\_spam)

good\_rate = 1.0 \* detec\_good / (detec\_good + non\_detec\_good)

accuracy = 1.0 \* (detec\_spam + detec\_good) / (len(prediction))

print("Total spam email")

print(total\_spam)

print("Total good email")

print(total\_good)

print("Spam rate is")

print(spam\_rate)

print("Good rate is")

print(good\_rate)

print("Accuracy")

print(accuracy)

Spam\_email\_filter\_LSTMs.py

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Sat Nov 4 15:01:46 2017

@author: lixiaodan

"""

import numpy as np

import email

import html2text

import re

import os

import NLP\_module

import collections

from nltk.corpus import stopwords

from keras.preprocessing import sequence

import LSTMSentenceClassifier

from keras.models import Sequential

from keras.layers import LSTM, Dense

from keras.layers.embeddings import Embedding

"""

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

Simple LSTM for Sequence Classification

"""

# fix random seed for reproducibility

np.random.seed(7)

# load the dataset but only keep the top n words, zero the rest

top\_words = 5000

# Build word vocabulary function

def build\_vocab(text, top\_words):

word\_counts = collections.Counter(text.split(' '))

word\_counts = word\_counts.most\_common(top\_words - 1)

vocab\_to\_ix\_dict = dict()

# Add unknown key --> 0 index

vocab\_to\_ix\_dict['unknown']=0

index = 1

for item in word\_counts:

vocab\_to\_ix\_dict[item[0]] = index

index = index + 1

# Create index --> vocab mapping

ix\_to\_vocab\_dict = {val:key for key,val in vocab\_to\_ix\_dict.items()}

return(ix\_to\_vocab\_dict, vocab\_to\_ix\_dict)

# Create a text cleaning function

def clean\_text(text\_string):

h = html2text.HTML2Text()

h.ignore\_links = True

h.escape\_snob = True

txt = h.handle(text\_string)

txt = txt.lower()

p = re.compile('\W+')

splits = p.split(txt)

result = ""

start\_to\_parse = False

for word in splits:

if word.find("\_") == 0:

word = word[1:]

if NLP\_module.strCmp(word, "Date"):

start\_to\_parse = True

if start\_to\_parse == False:

continue

found = re.search('[0-9]+', word)

if found != None:

continue

stop\_wds = stopwords.words("english")

stop\_wds.extend(["email", "www", "com", "http", "html", "gif", "smtp", "sender", "received", "zzzz", "yyyy","localhost", "org", "esmtp", "debian", "return", "path"])

if word in stop\_wds:

continue

if len(word) <= 2 or len(word) >= 10:

continue

result = result + " " + word

text\_string = result.lower()

return(text\_string)

labels = dict()

label\_path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/SPAMTrain.label'

infile = open(label\_path,'r')

label\_List = list()

for line in infile:

tp = line.split(" ")[1]

eml\_name = tp.split("\n")[0]

labels[eml\_name] = line.split(" ")[0]

label\_List.append(line.split(" ")[0])

infile.close()

path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/training\_new'

listing = os.listdir(path)

listing = listing

fail\_IO = list()

gd\_cnt = 0

bad\_cnt = 0

text\_target = list()

text\_data\_train = list()

texts = ""

for i in range(len(listing)):

fle = listing[i]

if str.lower(fle[-3:])=="eml":

try:

msg = email.message\_from\_file(open(path + '/' + fle))

strs = msg.as\_string()

cleantext = clean\_text(strs)

text\_data\_train.append(cleantext)

texts = texts + cleantext

if labels[fle] == "1":

gd\_cnt = gd\_cnt + 1

text\_target.append(1)

else:

bad\_cnt = bad\_cnt + 1

text\_target.append(0)

except UnicodeDecodeError:

fail\_IO.append(fle)

continue

min\_word\_freq = 100

max\_review\_length = 500

ix2word, word2ix = build\_vocab(texts, top\_words)

text\_processed = list()

# Convert text to word vectors

for s\_text in text\_data\_train:

s\_text\_words = s\_text.split(' ')

s\_text\_ix = list()

for ix, x in enumerate(s\_text\_words):

try:

s\_text\_ix.append(word2ix[x])

except:

s\_text\_ix.append(0)

cur\_text\_ix = s\_text\_ix[0:max\_review\_length]

if len(cur\_text\_ix) < 500:

for i in range(500 - len(cur\_text\_ix)):

cur\_text\_ix.append(0)

text\_processed.append(cur\_text\_ix)

# Shuffle and split data

text\_processed = np.array(text\_processed)

text\_data\_target = np.array(text\_target)

shuffled\_ix = np.random.permutation(np.arange(len(text\_data\_target)))

x\_shuffled = text\_processed[shuffled\_ix]

y\_shuffled = text\_data\_target[shuffled\_ix]

# Split train/test set

ix\_cutoff = int(len(y\_shuffled)\*0.80)

X\_train, X\_test = x\_shuffled[:ix\_cutoff], x\_shuffled[ix\_cutoff:]

y\_train, y\_test = y\_shuffled[:ix\_cutoff], y\_shuffled[ix\_cutoff:]

print("80-20 Train Test split: {:d} -- {:d}".format(len(y\_train), len(y\_test)))

# truncate and pad input sequences

max\_review\_length = 500

X\_train = sequence.pad\_sequences(X\_train, maxlen=max\_review\_length)

X\_test = sequence.pad\_sequences(X\_test, maxlen=max\_review\_length)

embedding\_vector\_length = 32

numHidden = 100

epochs = 40 # 40

batch\_size = 64

dropoutRate = 0.2

CNN\_model = LSTMSentenceClassifier.CNN\_Sentence\_Classifier(dropoutRate, top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train)

scores = CNN\_model.evaluate(X\_test, y\_test, batch\_size)

print("CNN Accuracy: %.2f%%" % (scores[1]\*100))

res\_CNN = CNN\_model.predict(X\_test)

binCNN = np.zeros((len(res\_CNN), 1), dtype=np.int)

for k in range(len(res\_CNN)):

if res\_CNN[k, 0] >= 0.5:

binCNN[k, 0] = 1

NLP\_module.plotRoc(binCNN, y\_test)

timesteps = 10

epochs = 40

print("\n The result for regular LSTM classifier")

regular\_LSTM = LSTMSentenceClassifier.LSTM\_Sentence\_Classifier(numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train)

result\_regular = regular\_LSTM.predict(X\_test)

binRegular = np.zeros((len(result\_regular), 1), dtype=np.int)

for i in range(len(result\_regular)):

if result\_regular[i,0] >= 0.5:

binRegular[i,0] = 1

NLP\_module.plotRoc(binRegular, y\_test)

scores = regular\_LSTM.evaluate(X\_test, y\_test, verbose=0)

print("Regular LSTM Accuracy: %.2f%%" % (scores[1]\*100))

print("\n The result for LSTM CNN classifier")

LSTM\_CNN = LSTMSentenceClassifier.CNN\_LSTM\_Sentence\_Classifier(numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train)

result\_LSTM\_CNN = LSTM\_CNN.predict(X\_test)

binLSTMCNN = np.zeros((len(result\_LSTM\_CNN), 1), dtype=np.int)

for i in range(len(result\_LSTM\_CNN)):

if result\_LSTM\_CNN[i,0] >= 0.5:

binLSTMCNN[i,0] = 1

NLP\_module.plotRoc(binLSTMCNN, y\_test)

scores = LSTM\_CNN.evaluate(X\_test, y\_test, verbose=0)

print("LSTM\_CNN Accuracy: %.2f%%" % (scores[1]\*100))

print("\n The result for LSTM model with dropout")

LSTM\_Dropout = LSTMSentenceClassifier.LSTM\_Dropout\_Sentence\_Classifier(dropoutRate, numHidden, top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train)

result\_LSTM\_dropout = LSTM\_Dropout.predict(X\_test)

binLSTMDrop = np.zeros((len(result\_LSTM\_dropout), 1), dtype=np.int)

for i in range(len(result\_LSTM\_dropout)):

if result\_LSTM\_dropout[i,0] >= 0.5:

binLSTMDrop[i,0] = 1

NLP\_module.plotRoc(binLSTMDrop, y\_test)

scores = LSTM\_Dropout.evaluate(X\_test, y\_test, verbose=0)

print("LSTM\_dropout Accuracy: %.2f%%" % (scores[1]\*100))

timesteps = 10

stacked\_LSTM = LSTMSentenceClassifier.Stacked\_LSTM\_Sentence\_Classifier(timesteps, numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train)

res\_stacked\_LSTM = stacked\_LSTM.predict(X\_test)

scores = stacked\_LSTM.evaluate(X\_test, y\_test, verbose=0)

print("Stacked LSTM Accuracy: %.2f%%" % (scores[1]\*100))

binStack = np.zeros((len(res\_stacked\_LSTM), 1), dtype=np.int)

for i in range(len(binStack)):

if res\_stacked\_LSTM[i,0] >= 0.5:

binStack[i, 0] = 1

NLP\_module.plotRoc(binStack, y\_test)

LSTMSentenceClassifier.py

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Mon Nov 6 15:54:55 2017

@author: lixiaodan

"""

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import LSTM

from keras.layers import Dropout

from keras.layers.convolutional import Conv1D

from keras.layers.convolutional import MaxPooling1D

from keras.layers.embeddings import Embedding

from keras.layers import GlobalAveragePooling1D

"""

The code in this file is based on the following source code:

https://machinelearningmastery.com/sequence-classification-lstm-recurrent-neural-networks-python-keras/

"""

def LSTM\_Sentence\_Classifier(numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train):

model = Sequential()

model.add(Embedding(top\_words, embedding\_vector\_length, input\_length=max\_review\_length))

model.add(LSTM(numHidden))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

model.fit(X\_train, y\_train, batch\_size, epochs)

return model

def LSTM\_Dropout\_Sentence\_Classifier(dropoutRate, numHidden, top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train):

model = Sequential()

model.add(Embedding(top\_words, embedding\_vector\_length, input\_length=max\_review\_length))

model.add(Dropout(dropoutRate))

model.add(LSTM(numHidden))

model.add(Dropout(dropoutRate))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

model.fit(X\_train, y\_train, batch\_size, epochs)

return model

def CNN\_LSTM\_Sentence\_Classifier(numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train):

# create the model

model = Sequential()

model.add(Embedding(top\_words, embedding\_vector\_length, input\_length=max\_review\_length))

model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))

model.add(MaxPooling1D(pool\_size=2))

model.add(LSTM(numHidden))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

model.fit(X\_train, y\_train, batch\_size, epochs)

return model

def CNN\_Sentence\_Classifier(dropoutRate, top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train):

seq\_length = max\_review\_length

model = Sequential()

model.add(Embedding(top\_words, embedding\_vector\_length, input\_length=max\_review\_length))

model.add(Conv1D(64, 3, activation='relu', padding='same', input\_shape=(seq\_length, 100)))

model.add(Conv1D(64, 3, activation='relu'))

model.add(MaxPooling1D(3))

model.add(Conv1D(128, 3, activation='relu'))

model.add(Conv1D(128, 3, activation='relu'))

model.add(GlobalAveragePooling1D())

model.add(Dropout(dropoutRate))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

print(model.summary())

model.fit(X\_train, y\_train, batch\_size=batch\_size, epochs=epochs)

return model

def Stacked\_LSTM\_Sentence\_Classifier(timesteps, numHidden,top\_words, embedding\_vector\_length, max\_review\_length, epochs, batch\_size, X\_train, y\_train):

model = Sequential()

model.add(Embedding(top\_words, embedding\_vector\_length, input\_length=max\_review\_length))

model.add(LSTM(numHidden, return\_sequences=True,

input\_shape=(timesteps, max\_review\_length))) # returns a sequence of vectors of dimension 32

model.add(LSTM(numHidden, return\_sequences=True)) # returns a sequence of vectors of dimension 32

model.add(LSTM(numHidden)) # return a single vector of dimension 32

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])

print(model.summary())

model.fit(X\_train, y\_train,

batch\_size, epochs)

return model

Naïve\_Bayes\_email\_spam.py

#!/usr/bin/env python3

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct 19 21:18:28 2017

@author: lixiaodan

"""

import email

import os

import re

import html2text

from nltk.corpus import stopwords

import math

import NLP\_module

import ast

"""

Method

Removing the following tokens from the vocabulary:

punctuations and numbers

english stopwords

one and two letter words

"""

def parse(words\_table, strings):

h = html2text.HTML2Text()

txt = h.handle(strings)

txt = txt.lower()

p = re.compile('\W+')

splits = p.split(txt)

for word in splits:

found = re.search('[0-9]+', word)

if found != None:

continue

stop\_wds = stopwords.words("english")

stop\_wds.extend(["email", "www", "com", "http", "html", "gif"])

if word in stop\_wds:

continue

if len(word) <= 2:

continue

if word in words\_table.keys():

words\_table[word] = words\_table.get(word) + 1

else:

words\_table[word] = 1

def processTable(words\_table):

total\_size = 0

dict\_after\_process = dict()

for word in words\_table.keys():

total\_size = total\_size + words\_table.get(word)

for word in words\_table.keys():

found = re.search('\_\_', word)

if found != None:

continue

if words\_table[word] >= 5:

dict\_after\_process[word] = words\_table[word]

return dict\_after\_process

labels = dict()

label\_path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/SPAMTrain.label'

infile = open(label\_path,'r')

label\_List = list()

for line in infile:

tp = line.split(" ")[1]

eml\_name = tp.split("\n")[0]

labels[eml\_name] = line.split(" ")[0]

label\_List.append(line.split(" ")[0])

infile.close()

path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/training\_new'

listing = os.listdir(path)

fail\_IO = list()

words\_table = dict()

gd\_table = dict()

bad\_table = dict()

gd\_cnt = 0

bad\_cnt = 0

for i in range(len(listing)):

fle = listing[i]

if str.lower(fle[-3:])=="eml":

try:

msg = email.message\_from\_file(open(path + '/' + fle))

strs = msg.as\_string()

if labels[fle] == "1":

gd\_cnt = gd\_cnt + 1

parse(gd\_table, strs)

gd\_table = processTable(gd\_table)

else:

bad\_cnt = bad\_cnt + 1

parse(bad\_table, strs)

bad\_table = processTable(bad\_table)

except UnicodeDecodeError:

fail\_IO.append(fle)

continue

# merge good and bad tables

good\_keys = list(gd\_table.keys())

bad\_keys = list(bad\_table.keys())

all\_words = good\_keys + bad\_keys

all\_words = set(all\_words)

all\_words\_list = list(all\_words)

for i in range(len(all\_words)):

curWord = all\_words\_list[i]

cnt1 = 0

cnt2 = 0

if curWord in gd\_table.keys():

cnt1 = gd\_table.get(curWord)

if curWord in bad\_table.keys():

cnt2 = bad\_table.get(curWord)

curcnt = cnt1 + cnt2

words\_table[curWord] = curcnt

# get prior probablity

total\_email = gd\_cnt + bad\_cnt

pGood = gd\_cnt \* 1.0 / total\_email

pBad = bad\_cnt \* 1.0 / total\_email

# get likelihood for bad words

v = len(words\_table.keys())

total\_bad = 0

poster\_bad = dict()

for bad\_wd in bad\_table.keys():

total\_bad = total\_bad + bad\_table.get(bad\_wd)

for bad\_wd in bad\_table.keys():

poster\_bad[bad\_wd] = (bad\_table.get(bad\_wd) + 1.0) / (total\_bad + v)

# get likelihood for good words

total\_gd = 0

poster\_gd = dict()

for gd\_wd in gd\_table.keys():

total\_gd = total\_gd + gd\_table.get(gd\_wd)

for gd\_wd in gd\_table.keys():

poster\_gd[gd\_wd] = (gd\_table.get(gd\_wd) + 1.0) / (total\_gd + v)

# The test case and get the posterior probability

test\_path = '/Users/lixiaodan/Desktop/ece590/CSDMC2010\_SPAM/CSDMC2010\_SPAM/test\_new'

tests = os.listdir(test\_path)

result = list()

predict = list()

test\_labels = list()

##1 stands for a HAM and 0 stands for a SPAM

for i in range(len(tests)):

fle = tests[i]

if str.lower(fle[-3:])=="eml":

cur\_table = dict()

try:

msg = email.message\_from\_file(open(test\_path + '/' + fle))

strs = msg.as\_string()

parse(cur\_table, strs)

cur\_table = processTable(cur\_table)

proBad = 1

proGd = 1

######## get the bad posterior probability for email

for wd in cur\_table.keys():

if wd in poster\_bad.keys():

proBad = proBad \* math.pow(poster\_bad[wd], cur\_table[wd])

else:

proBad = proBad \* math.pow((1.0 / (total\_bad + v)), cur\_table[wd])

proBad = proBad \* pBad

######## get the good posterior probability for email

for wd in cur\_table.keys():

if wd in poster\_gd.keys():

proGd = proGd \* math.pow(poster\_gd[wd], cur\_table[wd])

else:

proGd = proGd \* math.pow((1.0 / (total\_gd + v)), cur\_table[wd])

proGd = proGd \* pGood

pair = list()

pair.append(fle)

if proBad > proGd:

pair.append(0)

predict.append(0)

else:

pair.append(1)

predict.append(1)

test\_labels.append(ast.literal\_eval(labels[fle]))

result.append(pair)

except UnicodeDecodeError:

fail\_IO.append(fle)

continue

NLP\_module.plotRoc(predict, test\_labels)

detec\_spam = 0

tt\_spam = 0

detec\_good = 0

tt\_good = 0

for i in range(len(predict)):

if test\_labels[i] == 0:

tt\_spam = tt\_spam + 1

if predict[i] == 0:

detec\_spam = detec\_spam + 1

if test\_labels[i] == 1:

tt\_good = tt\_good + 1

if predict[i] == 1:

detec\_good = detec\_good + 1

spam\_rate = 1.0 \* detec\_spam / tt\_spam

good\_rate = 1.0 \* detec\_good / tt\_good

accuracy = 1.0 \* (detec\_spam + detec\_good) / (tt\_spam + tt\_good)