# Text Message Classification: Model to Determine Spam or Real Texts

## John Figueroa & Megan Jennings

### Introduction

Over the past decade, sending text messages through electronic communications has exploded. It has become a primary means of communication. With this explosion, there has also been a sharp increase in unwanted text messages that individuals have received, or spam. Individuals have found these to be very bothersome. Through the data, how can one determine what is spam and what is a real message? This is where a text classification model is needed.

MonkeyLearn.com provides a clear, precise definition of text classification. From their website, “Text classification is the process of assigning tags or categories to text according to its content. It’s one of the fundamental tasks in Natural Language Processing (NLP) with broad applications such as sentiment analysis, topic labeling, spam detection, and intent detection.”[[1]](#footnote-2) Some examples of text classification use are: grouping articles by topic, providing a brief summary of a document, or determining the importance of an email. In general terms, text classification is the process of identifying tags from unstructured data & using the tags as categories.

The purpose of this project is to build a model that can classify text messages as either spam or ham (non-spam). The two methods used for comparison were classic Logistic Regression and deep learning LSTM (Long Short-Term Memory). Long short-term memory (LSTM) is an artificial recurrent neural network, (RNN) architecture. Logistic Regression requires manual feature engineering; whereas, LSTM is a deep learning methodology that allows the model to determine the optimal features. While both methods generate models with high accuracy, there is a lot of manual effort needed to determine the features needed for high accuracy in the logistic regression method.

### The Data

The SMS spam dataset used in the project is a public dataset that has been collected for mobile phone spam research. The dataset contains 5,574 English, real and non-encoded messages that were labeled as ham or spam. The data set was comprised of data collected from 4 internet sources that are free for research. The full details on the corpus can be accessed via this webpage, <http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>.

Figure 1 is representative of the data prior to any modifications or exclusions. Specifically, the data is comprised of fields labeled v1, v2, v3, v4, & v5. Upon inspection of the data, it was found that v1 represented the label (ham or spam). The field v2 represented the body of the text message (i.e., the message itself). The remaining fields (v3, v4, & v5) were mostly empty (Out[4] in Figure 1) & include portions of the same text that is in v2.

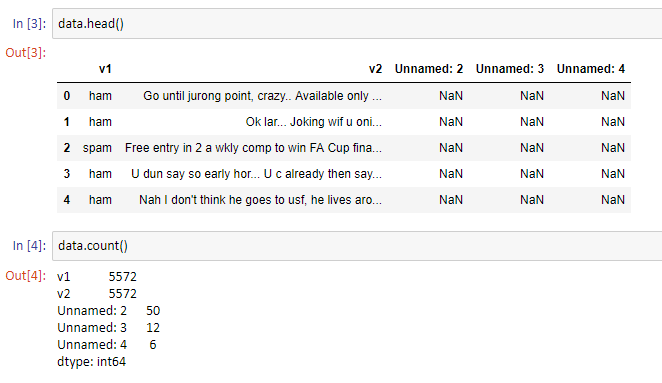


Figure : Example Rows of Raw Data

### Data Processing

As with any dataset, the key to providing valuable insights is to understand the data that is being used for modeling. The Pandas library within python has many functions that allow researchers to explore their data. The first step is to clean up the data so the insights you make are valuable. Fields v3, v4, & v5 were removed due to the large count of missing values. After that, the words ham & spam were assigned numeric representations of themselves (0 for ham & 1 for spam). Following the re-mapping, a pie chart ofthe data was created to get the proportions of ham to spam. Of the 5,574 text messages that comprise our dataset, 747 (~13%) were spam and 4,827 (~87%) were ham (real). This proportion can be seen in Figure 2.

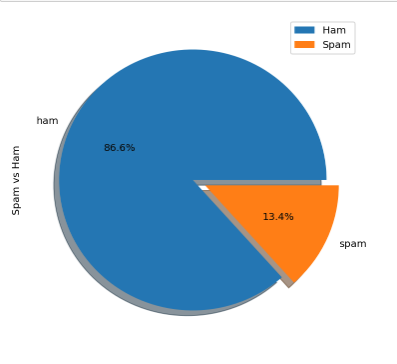
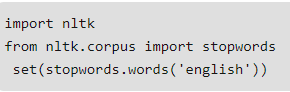


Figure 2: Porportion Representation of Ham/Spam Data

The next step was text processing. This includes removing stop words & tokenizing the messages. Stop words are words that do not typically add to the context of text. There is not a universal list of stop words. For this project, we used the nltk.corpus python library’s built in list. Examples of the stop words removed, include ‘is’, ‘a’, and ‘the’. A full list of Stop Words can be found by running the following block of code in a python shell.



Tokenizing is the process of breaking up a sequence of text into pieces such as words, keywords, phrases, symbols and other elements called tokens. For the purposes of this project, we tokenized the messages into individual words.

Next, a function was created to identify the most common types of words for each message type. Word Clouds for each type were built to get a visual representation of the primary words within each sub-group. These word clouds can be seen in figures 3 & 4.

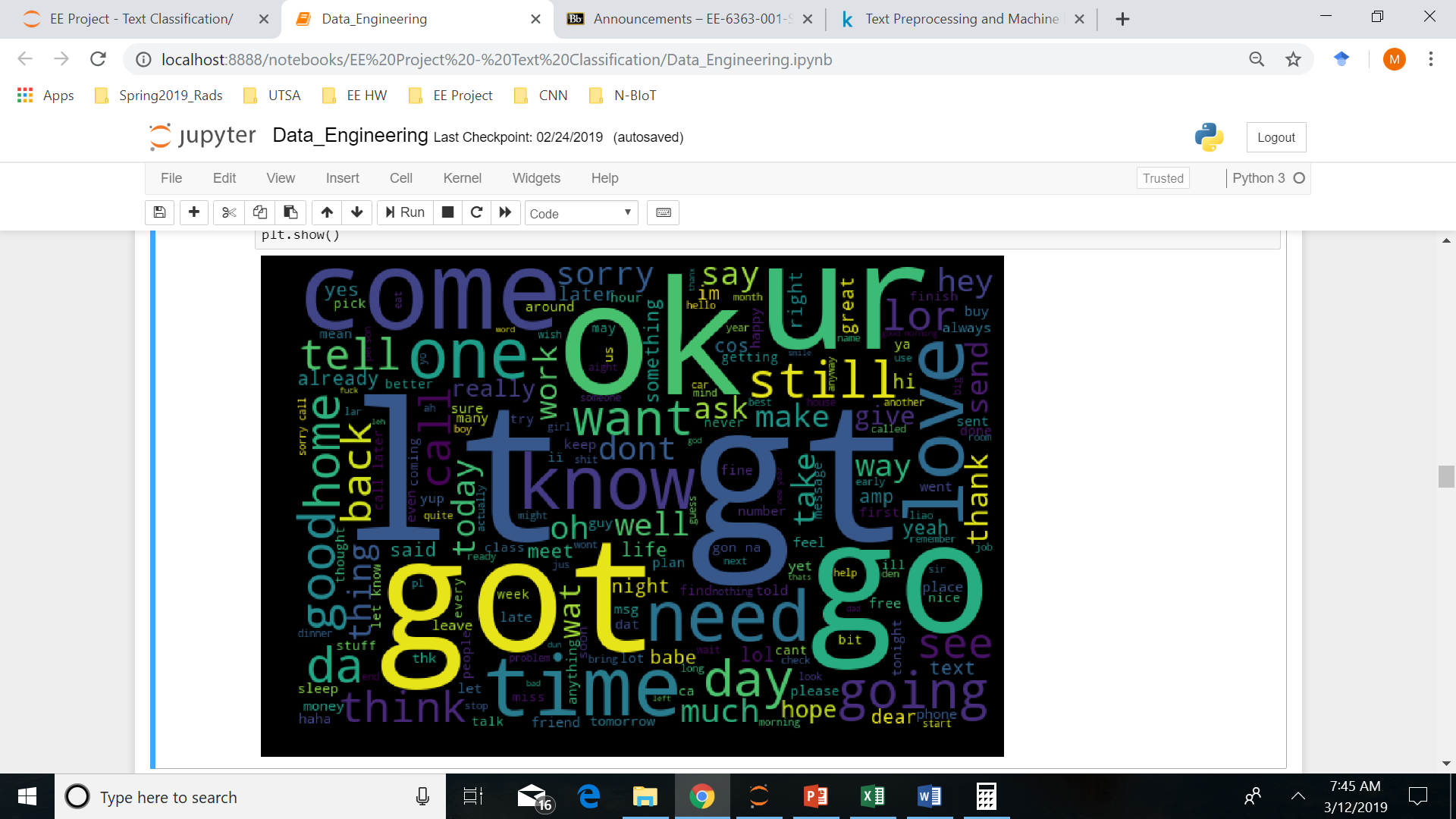
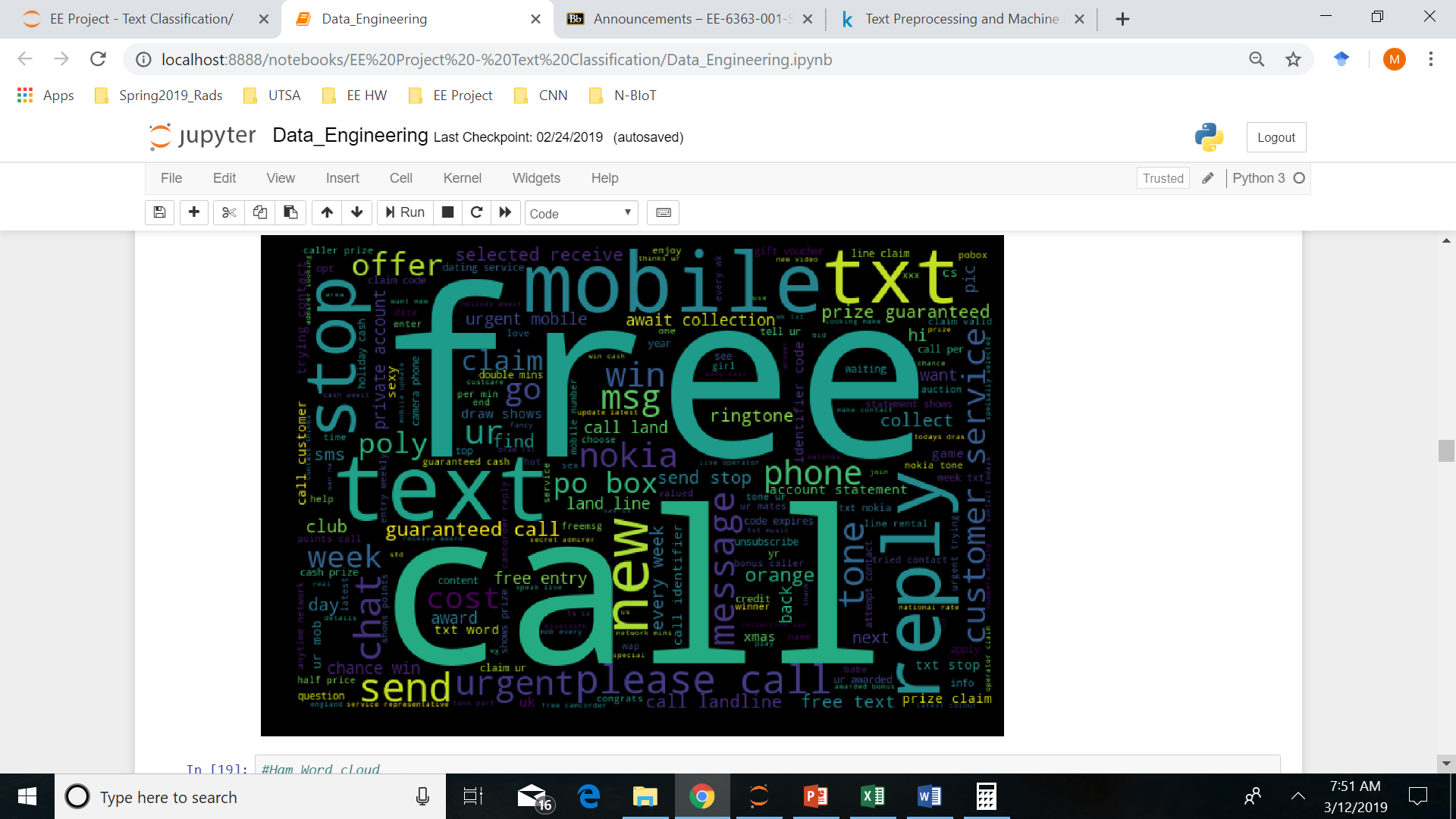
 

Figure 3: Word Cloud for ham messages Figure 4: Word Cloud for spam messages

Next, we investigated what types of features may be indicative of ham vs spam. We investigated message length & found that spam messages were typically longer than ham messages as can be seen in figure 5.

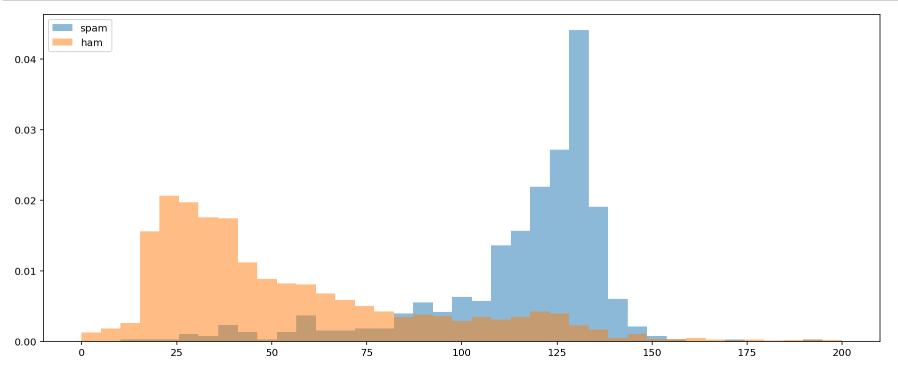


Figure 5: Length of Ham vs Spam

Another metric that differed between ham & spam were the number of numeric values in each dataset. Long Numbers is a metric that represents the number of digits beyond 7 digits long. Short Number represents the number of numeric values in the message that are less than 7 digits long. These comparisons can be seen in Figures 6 & 7.

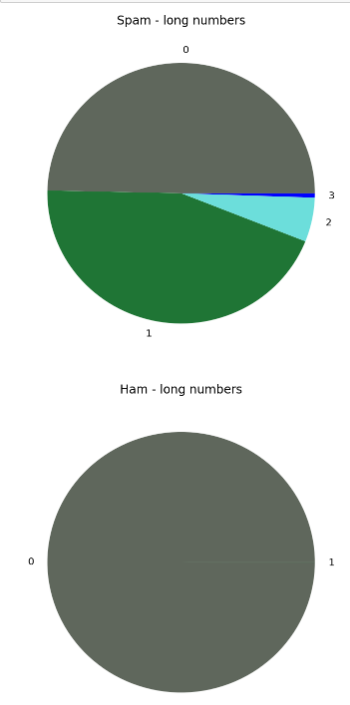
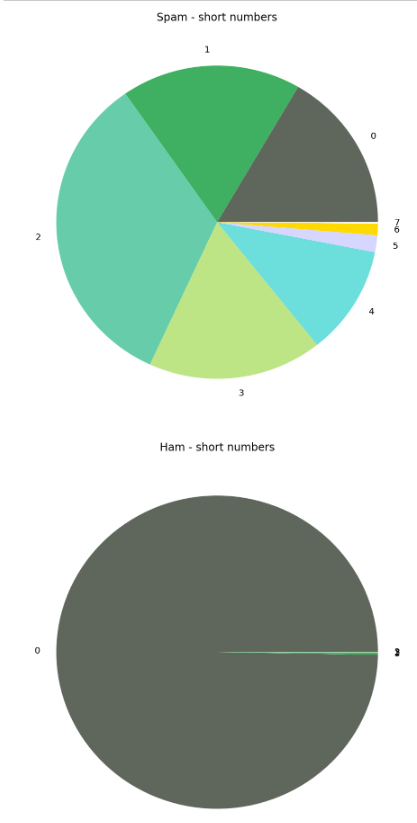
 

Figure 6: Short Numbers Figure 7: Long Numbers

In addition to the functions based on length, other functions were created to represent the amount of punctuation in each message as well as flag yes or no if a website was included in the message. For the website, it was found that if a ham message included a website it typically was just the domainname.com (for example, amazon.com); where as spam messages included full website names with https:// & www included. These parameters were used to flag if a website was included in the text message. Figure 8 is representative of the dataset after the new metrics were added.

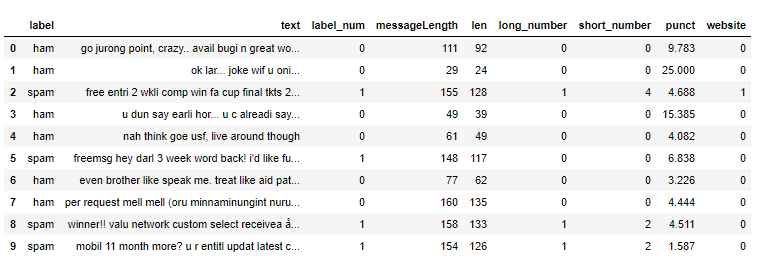


Figure 8: Dataset with new features added

Now, the models are ready to be built.

### Splitting the Data

Outlined in the Data Processing section, the disproportion of ham to spam caused stratified sampling to be used to avoid the training models being skewed towards ham. Stratified sampling is a sampling method that allows the researcher to gain insights on different sub-groups. In our case the sub groups are ham & spam. The python library scikit has a stratify option within the model\_selection train\_test\_split function. By specifying the label as the parameter of the function, the function makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter. An 80/20 split was applied to the dataset with the stratified sampling to generate the test and train data for the models that ensured 13% spam & 87% ham in each of test & train.

### Classic Modeling

To get a feel for the benefit of deep learning, several models were created based on the modified dataset (i.e., we did the feature engineering instead of letting the computer). We used random forest, Multinomial Bayes, and Logistic Regression. Logistic Regression was the best performer. The final model had only 3 ham messages incorrectly labeled as spam from the entire dataset. There were 23 spam messages incorrectly labeled as ham. The result was expected as this hypothesis would rather let spam through than miss a ham message. The project would much rather have false negatives (type 2) than false positives (type 1). The confusion matrix & corresponding heat map are seen in Figure 9.

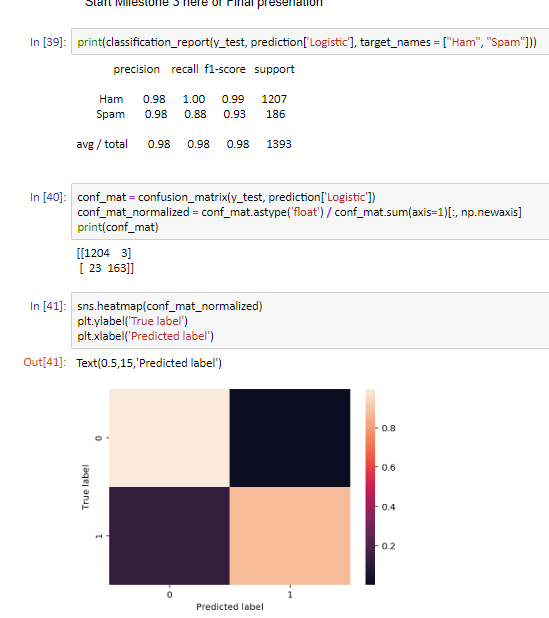


Figure 9: Results of Logistic Model

### Machine Learning

The purpose of machine learning is to let the computer learn the features that best predict our desired outcome. Text classification is best handled by a Recurrent Neural Network (RNN). RNN is preferred over CNN in this case to address the issue of vanishing gradient. In a multi-layer network, gradients for deeper layers are calculated as products of many gradients (of activation functions). When those gradients are small or zero, it will easily vanish. So, it becomes very hard to calculate and update. There are two factors that affect the magnitude of gradients - the weights and the activation functions (or more precisely, their derivatives) that the gradient passes through. If either of these factors is smaller than 1, then the gradients may vanish in time. The LSTM architecture addresses this problem by introducing a memory cell that can preserve state over long periods of time. Here, by changing the values of the memory cell one can control the amount of information the network retains/discards over the entire input series as well as control dependency on individual inputs. The increased regulation helps one overcome the vanishing/ exploding gradient problem. Assigning memory to each cell and editing it as new input are given, helps one retain dependencies from earlier inputs and conserve state over long training sessions.

For our research, we used the RNN architecture known as LSTM (Long short-term memory). Similar steps were taken to clean the data as we discussed in Data Processing section above. However, for Machine Learning we do not create new features so only the text processing was completed. Specifically, stop words were removed, used a single word tokenizer, and padded the sequences so they were all the same length. In order to batch the LSTM, the inputs must all be the same length. Since we saw the various lengths in out data exploration, padding was used to force all the messages to be the same input length. The same stratified sampling discussed in the ‘Splitting the Data’ section was used for the LSTM Model.

The model structure for our LSTM can be seen in Figure 10. Our model consists of 1 embedding layer, an LSTM layer, 1 fully connected layer, a 50% dropout layer, & 1 final fully connected output layer.

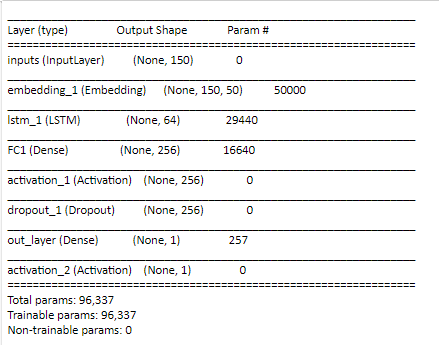


Figure 10: LSTM Structure

For compiling the model, the binary cross entropy loss function was chosen. Cross-entropy loss, or log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. Cross-entropy loss increases as the predicted probability diverges from the actual label.

The RMSprop optimizer was used. The RMSprop optimizer is like the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. Model selection was based on accuracy in predicting Spam vs Ham.

In order to fit our model, the training data set was again split 80/ 20 into training & validation. The Validation accuracy achieved after 10 epochs was 99.84%. This shows the model performs well & this configuration is chosen as the final model. The model can now run on the test data as seen in Figure 11. Evaluating the model on the test data provides 98.5% accuracy & translates as being a very good model for predicting ham vs spam in text message classification.

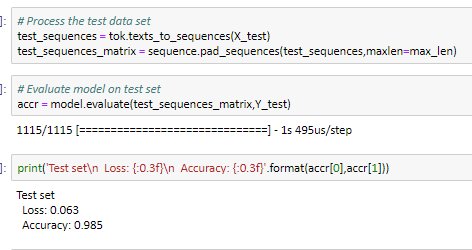
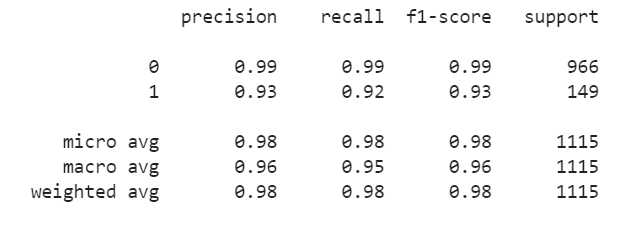


Figure 11: Evaluation of model on test data

Additionally, as mentioned in the Logistic Regression section, the primary metric for picking a model is accuracy combined with precision. The precision for our selected model is 99% for Ham & 93% for Spam.



### Conclusion

The benefit of the RNN over the traditional Logistic Regression method was in the effort it took to find the features & determine which were relevant & which were not in the classic method. The RNN determined the features that were important & applied them in a way that allowed just as much accuracy with less manual effort. Additionally, the RNN is better suited to being used in transfer learning & in generalizing the results to other data sets.

## Appendix

### References

Dataset:  <http://www.dt.fee.unicamp.br/~tiago/smsspamcollection/>

Acknowledgement: <https://www.kaggle.com/uciml/sms-spam-collection-dataset>

### Full Code

The code used in this project with outputs can be found here: <https://github.com/msjennings/EE-6363-Deep-Learning-Project/tree/master>

#### LSTM Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import LabelEncoder

from keras.models import Model

from keras.layers import LSTM, Activation, Dense, Dropout, Input, Embedding

from keras.optimizers import RMSprop

from keras.preprocessing.text import Tokenizer

from keras.preprocessing import sequence

from keras.utils import to\_categorical

from keras.callbacks import EarlyStopping

%matplotlib inline

df = pd.read\_csv('spam.csv',delimiter=',',encoding='latin-1')

df.head()

df.drop(['Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'],axis=1,inplace=True)

df.info()

X = df.v2

Y = df.v1

le = LabelEncoder()

Y = le.fit\_transform(Y)

Y = Y.reshape(-1,1)

X\_train,X\_test,Y\_train,Y\_test = train\_test\_split(X,Y,test\_size=0.2, stratify=Y)

max\_words = 1000

max\_len = 150

tok = Tokenizer(num\_words=max\_words)

tok.fit\_on\_texts(X\_train)

sequences = tok.texts\_to\_sequences(X\_train)

sequences\_matrix = sequence.pad\_sequences(sequences,maxlen=max\_len)

def LSTM():

inputs = Input(name='inputs',shape=[max\_len])

layer = Embedding(max\_words,50,input\_length=max\_len)(inputs)

layer = LSTM(64)(layer)

layer = Dense(256,name='FC1')(layer)

layer = Activation('relu')(layer)

layer = Dropout(0.5)(layer)

layer = Dense(1,name='out\_layer')(layer)

layer = Activation('sigmoid')(layer)

model = Model(inputs=inputs,outputs=layer)

return model

model = LSTM()

model.summary()

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

model.fit(sequences\_matrix,Y\_train,batch\_size=128,epochs=10, validation\_data=(sequences\_matrix,Y\_train),

validation\_split=0.2,callbacks=[EarlyStopping(monitor='val\_loss',min\_delta=0.0001)])

test\_sequences = tok.texts\_to\_sequences(X\_test)

test\_sequences\_matrix = sequence.pad\_sequences(test\_sequences,maxlen=max\_len)

accr = model.evaluate(test\_sequences\_matrix,Y\_test

print('Test set\n Loss: {:0.3f}\n Accuracy: {:0.3f}'.format(accr[0],accr[1]))

from sklearn.metrics import classification\_report

y\_pred = model.predict(test\_sequences\_matrix)

print(classification\_report(Y\_test, y\_pred.round()))

#0 - Ham 1 - Spam

1. <https://monkeylearn.com/text-classification/> [↑](#footnote-ref-2)