Email & SMS Spam Word Cloud

Rayyan Kazim

2025-04-04

Introduction & Relevance

In a world and time where spam messages are becoming extremely common, it is important to know and understand how to avoid them. However, most individuals will not know how to avoid them when they won't be able to classify whether or not a message is spam in the first place. The main purpose of this project is to build models that can predict whether or not a message is spam, and to identify the words that are most commonly used in spam messages. This will help spread awareness when it comes to avoiding these types of messages.

This project contains three parts. In the first part, a Machine Learning model/algorithm will be used to analyze spams within Emails. The second part will consist of exploring real examples of SMS texts, and using a Neural Network (towards Natural Language Processing) to understand whether or not they are spam. This will allow an exploration of spams within both Emails and SMS texts. The third part (combining first two parts) will be about building a word cloud with frequent spam words in both Emails and SMS texts.

Firstly, Let's gather our libraries:

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

from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import neighbors
from sklearn import metrics
from sklearn.preprocessing import scale
from sklearn.feature_extraction.text import TfidfVectorizer

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input



from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import nltk
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from nltk.stem import WordNetLemmatizer
```

```
from wordcloud import WordCloud
```

Part 1: Spam Emails

We will consider the Spambase dataset from Kaggle (link: https://www.kaggle.com/datasets/colormap/spambase). This can also be found on the UC Irvine ML repository.

To analyze this dataset, we will consider and compare multiple algorithms and we will use the best one as our model for feature selection. We will compare a Neural Network (muli-layer perceptron classification) with two Ensemble methods (for boosting, we will use Gradient Boosting, and for bagging, we will use a Random Forest).

Exploratory Analysis

```
df = pd.read_csv("spambase.csv")
df.head(5)
```

	word_freq_make	word_freq_address	word_freq_all	word_freq_3d	word_freq_our	word_freq_over	word_freq_re
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31

```
df.shape
```

```
(4601, 58)
```

This dataset has 4601 observations with 58 columns where there is only one target column/variable, "spam". In this column, 1 implies "spam" and 0 implies "not spam". All of the other columns represent a unique word or symbol that may come up in the emails. Let's drop all columns that don't define a unique word. For the ones that do define a unique word, let's drop the "word_freq_" part of their name for simplicity.

```
df = df.drop(['char_freq_;', 'char_freq_(', 'char_freq_[', 'char_freq_!',
              'char_freq_$', 'char_freq_#', 'capital_run_length_average',
              'capital_run_length_longest', 'capital_run_length_total'], axis=1)
```

```
df.columns = df.columns.str.replace('word_freq_', '', regex=True)
df.head(5)
```

	make	address	all	3d	our	over	remove	internet	order	mail	...	direct	cs	meeting	original	project
0	0.00	0.64	0.64	0.0	0.32	0.00	0.00	0.00	0.00	0.00	...	0.00	0.0	0.0	0.00	0.0
1	0.21	0.28	0.50	0.0	0.14	0.28	0.21	0.07	0.00	0.94	...	0.00	0.0	0.0	0.00	0.0
2	0.06	0.00	0.71	0.0	1.23	0.19	0.19	0.12	0.64	0.25	...	0.06	0.0	0.0	0.12	0.0
3	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.0	0.0	0.00	0.0
4	0.00	0.00	0.00	0.0	0.63	0.00	0.31	0.63	0.31	0.63	...	0.00	0.0	0.0	0.00	0.0

```
df.spam.value_counts()
```

```
spam
0    2788
1    1813
Name: count, dtype: int64
```

Data Splitting, Testing & Training

Let's allow x to be a dataframe consisting of all predictor variables and y as the response variable. We will use 25% of our data for testing and the rest for training.

```
x = df.drop('spam', axis=1)
y = df['spam']

x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.25, random_state=1, stratify=y)
```

Classification Comparisions

```
Methods = {"Random Forest": RandomForestClassifier(),
           "Gradient Boosting": GradientBoostingClassifier(),
           "Neural Network": MLPClassifier(hidden_layer_sizes = (50,), activation='relu', max_iter=300)
           }
```

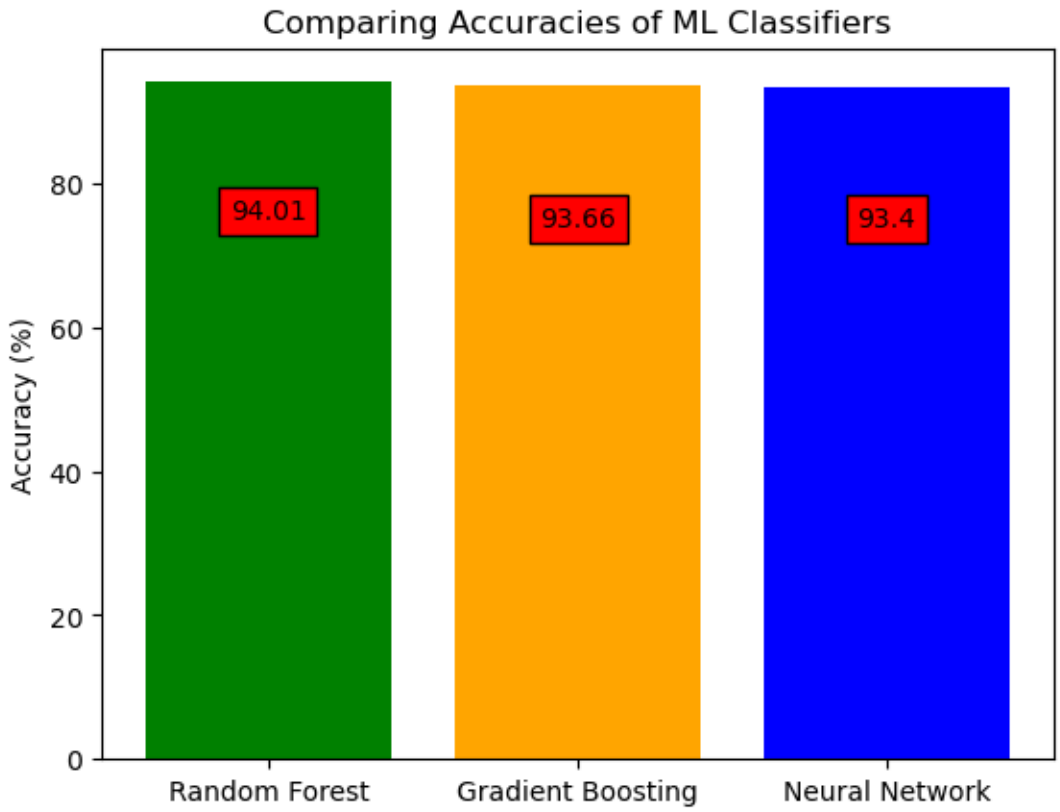
```
def accuracy(classifier, X_train, X_test, Y_train, Y_test):

    accuracy_scores = {}
```

```
    for Classifier, Algorithm in Methods.items():
        model = Algorithm.fit(X_train, Y_train)
        prediction = model.predict(X_test)
        accuracy_score = metrics.accuracy_score(Y_test, prediction)
        accuracy_score = np.round(accuracy_score*100, 2)

        accuracy_scores[Classifier] = accuracy_score

    return accuracy_scores

accuracies = accuracy(Methods, x_train, x_test, y_train, y_test)
```

```
/Applications/anaconda3/envs/data3ml3/lib/python3.9/site-packages/sklearn/neural_network/_multilayer_perceptron.py:
  warnings.warn(
```

```
def plot_labeling(predictor, response):
    for i in range(len(predictor)):
        plt.text(i, response[i] //1.25, response[i], horizontalalignment = 'center',
                 bbox = dict(facecolor='red'))

plt.bar(list(accuracies.keys()), list(accuracies.values()),
        color = ['green', 'orange', 'blue'])

plot_labeling(range(len(accuracies)), list(accuracies.values()))
plt.ylabel("Accuracy (%)")
plt.title("Comparing Accuracies of ML Classifiers")
```

```
Text(0.5, 1.0, 'Comparing Accuracies of ML Classifiers')
```



We can see that all three of our classifiers have fairly similar accuracies, and they are all extremely high (>90%). However, we will be moving forward with the Random Forest classifier as it has a slightly higher accuracy than the other classifiers. This can be viewed as a positive thing due to the fact that Random Forest is well know for being able to derive variable importances.

Feature Importance

```
def top_features(classifier, x, y, x_train):

    classifier.fit(x, y)
    feature_importances = classifier.feature_importances_
    sorted_indices = feature_importances.argsort()[::-1]
    feature_names = x_train.columns[sorted_indices]
    importances = feature_importances[sorted_indices]

    overall_importances = dict(zip(feature_names, importances))
    Top15 = dict(list(overall_importances.items())[:15])

    return Top15
```

```
MostImportantFeatures_Email = top_features(RandomForestClassifier(), x, y, x_train)
MostImportantFeatures_Email
```

```
{'free': 0.1038957656417889,
 'remove': 0.09825727236222631,
 'your': 0.08302107381825603,
 'money': 0.06593772233716057,
 'hp': 0.05636764644126155,
 'our': 0.0522682294429695,
 '000': 0.05151681460188087,
 'you': 0.050047425649101064,
 'george': 0.03465411472237114,
 'edu': 0.026537094581931958,
 're': 0.02557219116933251,
 'internet': 0.025337669056769804,
 'business': 0.025251624364215464,
 'all': 0.022991979449127516,
 'hpl': 0.02150218273515626}
```

The function above gives us the 15 most important variables/words in predicting whether or not Emails are Spams. Given that this dataset started with over 50 variables, extracting the 15 most important ones is reasonable for the highest feature importances.

Below is a list of the 15 most important Spam words within Spam Emails. We will come to this later.

```
Spam_Words_Emails = list(MostImportantFeatures_Email.keys())
```

Part 2: Spam SMS Texts

Now, we will view/explore examples of real SMS messages where some are classified as spam, and some are not. The following text file, "SMSSpamCollection" has been obtained through the UC Irvine ML repository (Link: https://archive.ics.uci.edu/dataset/228/sms+spam+collection).

We will first convert the TSV file into a dataframe, so that we have one column with messages, and one target column which is able to classify whether or not a message is spam.

Data Overview

```
df2 = pd.read_csv("SMSSpamCollection", sep='\t', names=['label', 'messages'])
df2.head(4)
```

	label	messages
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...

```
df2.shape
```

```
(5572, 2)
```

We have exactly 5572 SMS messages within our data.

Let's label our spam messages as "1", and the non-spam messages as "0", for ML purposes.

```
df2.label = df2["label"].replace("ham", 0)
```

```
df2.label = df2["label"].replace("spam", 1)
df2.head(4)
```

```
/var/folders/z6/4k03kssx2hsf7f1vh9g6w4gm0000gn/T/ipykernel_94725/1434396293.py:1: FutureWarning: Downcasting behavi
  df2.label = df2["label"].replace("spam", 1)
```

	label	messages
0	0	Go until jurong point, crazy.. Available only ...
1	0	Ok lar... Joking wif u oni...
2	1	Free entry in 2 a wkly comp to win FA Cup fina...
3	0	U dun say so early hor... U c already then say...

Data Splitting, Testing & Training

Similarly to the previous section, we will split our data into a predictor variable (x2) and a response variable (y2). Again, we will use 25% of our data for testing and the remaining 75% for training.

```
x2 = df2["messages"]
y2 = df2["label"]

x_train_2, x_test_2, y_train_2, y_test_2 = train_test_split(
    x2, y2, test_size=0.25, random_state=1, stratify = y2)
```

Neural Network Classification (NLP Stage)

Important Note: For reference, lets use the following link:

https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html.

This link is helpful with understanding how to utilize Sklearn for text-based data.

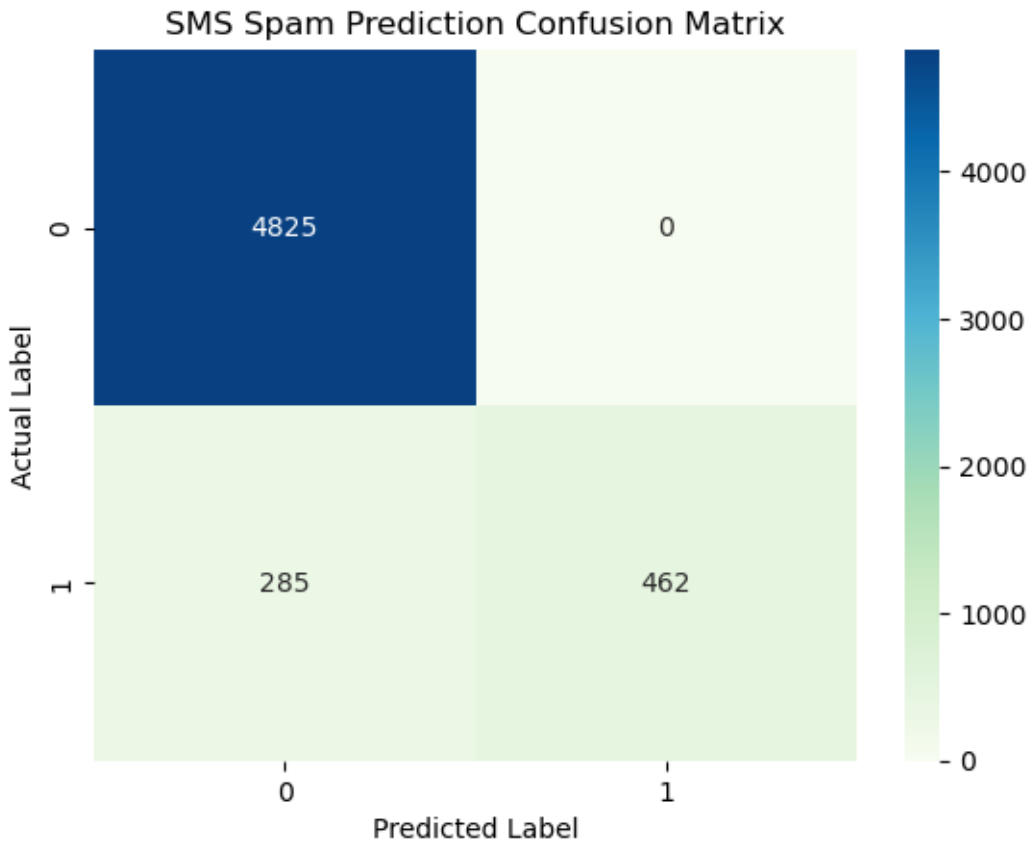
Because our dataframe consists of text values, for the purposes of Machine Learning, lets transform them into numerical values using a vectorize function. Becuase the vectorization of our text data will add a lot more features/predictors to the dataframe, We will construct a Neural Network with 50 neurons, and fit this model into our training sets. We will then use our fitted Neural Network to make predictions on our entire dataset to assess which messages are spam, and which ones are not.

```
vectorizer = TfidfVectorizer()
x_train_2_vectorized = vectorizer.fit_transform(x_train_2).toarray()
```

To allow our Neural Network to learn complex patterns, we will use the ReLU activation function in our hidden layer with 50 neurons. Because this is a binary classification problem, we will use the Sigmoid activation function in the network's output layer.

```
Neural_Network = Sequential([
    Input(shape = (x_train_2_vectorized.shape[1],)),
    Dense(50, activation = 'relu'),
    Dense(1, activation='sigmoid')
])

Neural_Network.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accuracy')



Neural_Network.fit(x_train_2_vectorized, y_train_2)

x2_vectorized = vectorizer.transform(x2)
predictions_SMS = Neural_Network.predict(x2_vectorized)
predictions_SMS = (predictions_SMS > 0.5).astype(int).flatten()



CM_NN = confusion_matrix(y2, predictions_SMS)
sns.heatmap(CM_NN, annot = True, fmt='d', cmap = 'GnBu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title("SMS Spam Prediction Confusion Matrix")
```

```
131/131 [==============================] - 0s 1ms/step - loss: 0.4375 - accuracy: 0.8751
175/175 [==============================] - 0s 418us/step

2025-10-05 21:13:12.808343: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executo
     [[{{node Placeholder/_0}}]]

Text(0.5, 1.0, 'SMS Spam Prediction Confusion Matrix')
```

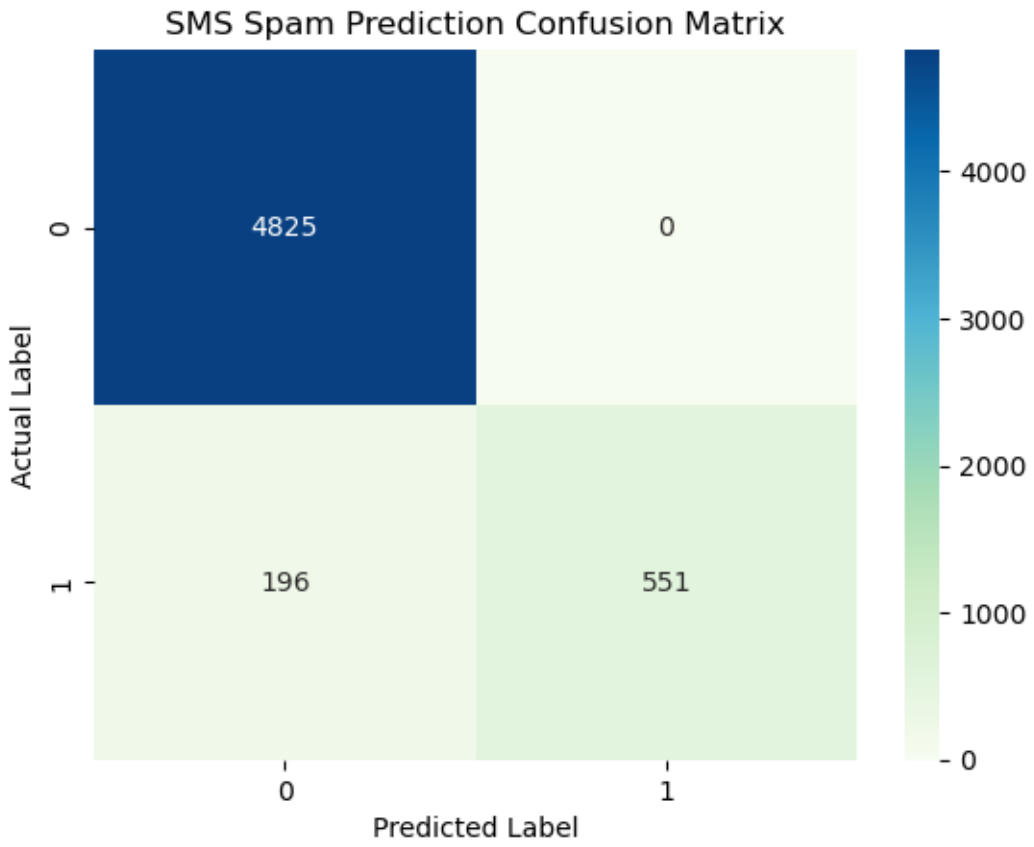


While our model performed well, there were still a significant amount of spam messages, that were classified as non-spam. To allow our Neural Network to perform a better classification task, we will increase the neuron count to 100.

```
Neural_Network_2 = Sequential([
    Input(shape = (x_train_2_vectorized.shape[1],)),
    Dense(100, activation = 'relu'),
    Dense(1, activation='sigmoid')
])

Neural_Network_2.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accuracy')
```

```
Neural_Network_2.fit(x_train_2_vectorized, y_train_2)

predictions_SMS = Neural_Network_2.predict(x2_vectorized)
predictions_SMS = (predictions_SMS > 0.5).astype(int).flatten()



CM_NN = confusion_matrix(y2, predictions_SMS)
sns.heatmap(CM_NN, annot = True, fmt='d', cmap = 'GnBu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title("SMS Spam Prediction Confusion Matrix")
```

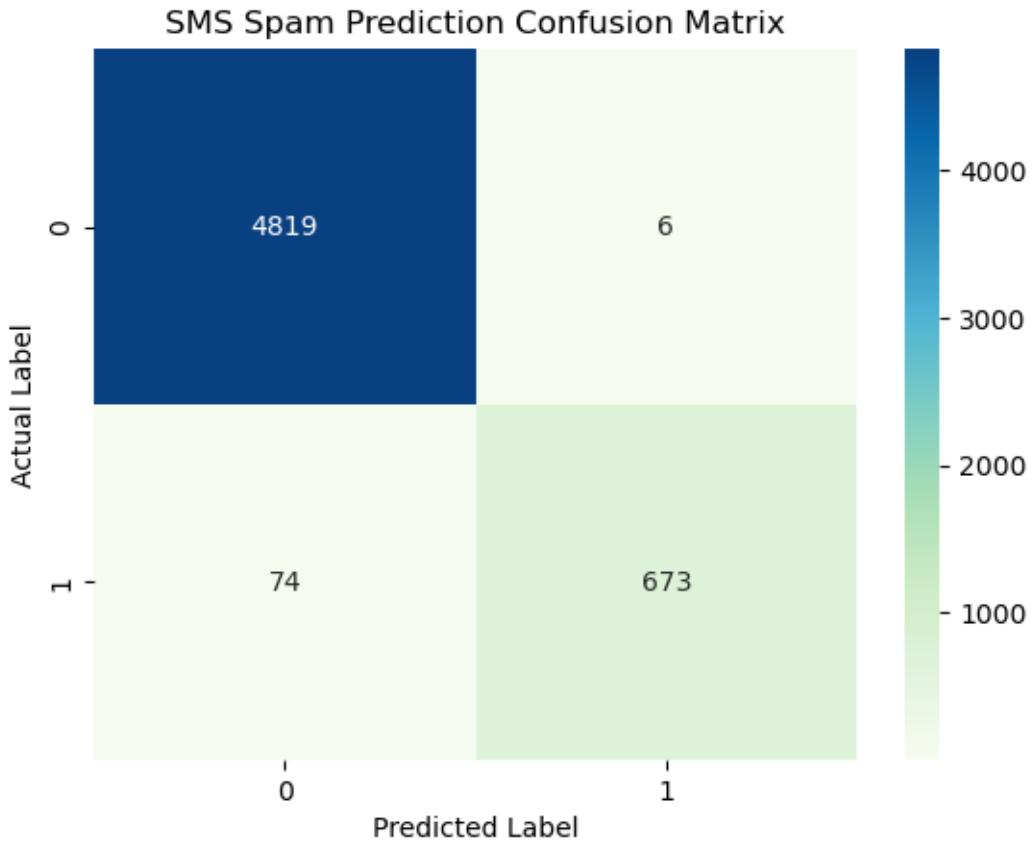
```
131/131 [==============================] - 0s 2ms/step - loss: 0.3737 - accuracy: 0.8863
175/175 [==============================] - 0s 388us/step

2025-10-05 21:21:41.219385: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executo
     [[{{node Placeholder/_0}}]]

Text(0.5, 1.0, 'SMS Spam Prediction Confusion Matrix')
```



With a clear improvement in performance from 50 to 100 neurons, we will increase our neuron count even further.

```
Neural_Network_3 = Sequential([
    Input(shape = (x_train_2_vectorized.shape[1],)),
    Dense(300, activation = 'relu'),
    Dense(1, activation='sigmoid')
])

Neural_Network_3.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = 'accuracy')



Neural_Network_3.fit(x_train_2_vectorized, y_train_2)

predictions_SMS = Neural_Network_3.predict(x2_vectorized)
predictions_SMS = (predictions_SMS > 0.5).astype(int).flatten()



CM_NN = confusion_matrix(y2, predictions_SMS)
sns.heatmap(CM_NN, annot = True, fmt='d', cmap = 'GnBu')
plt.xlabel('Predicted Label')
plt.ylabel('Actual Label')
plt.title("SMS Spam Prediction Confusion Matrix")
```

```
131/131 [==============================] - 1s 5ms/step - loss: 0.3079 - accuracy: 0.8973
175/175 [==============================] - 0s 588us/step

2025-10-05 21:24:45.151273: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executo
     [[{{node Placeholder/_0}}]]

Text(0.5, 1.0, 'SMS Spam Prediction Confusion Matrix')
```



Based on the Confusion Matrix above, it is clear that our Neural Network performed extremely well. It successfully predicted majority of the actual spam messages as spam, and non-spam messages as non-spam. Earlier, we saw that our dataframe has 5572 SMS messages. Out of those 5572 messages, only 80 were misclassified, whereas the rest were correctly classified (very good ratio). Now, lets add our predictions into our dataframe.

```
df2["predictions"] = predictions_SMS
```

Word Tokenization (NLP Stage)

Important Note: For reference, lets use the following link:

https://www.datacamp.com/tutorial/text-analytics-beginners-nltk

```
nltk.download('punkt')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     /Users/rayyankazim/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to
```

```
[nltk_data]     /Users/rayyankazim/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
True
```

```
import re
```

```
def tokenizer(text):
    return re.findall(r'\b\w+\b', text.lower())

df2['tokens'] = df2['messages'].apply(tokenizer)
```

```
df2.head(4)
```

	label	messages	predictions	tokens
0	0	Go until jurong point, crazy.. Available only ...	0	[go, until, jurong, point, crazy, available, o...
1	0	Ok lar... Joking wif u oni...	0	[ok, lar, joking, wif, u, oni]
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...
3	0	U dun say so early hor... U c already then say...	0	[u, dun, say, so, early, hor, u, c, already, t...

```
english_stopwords = set(stopwords.words('english'))

# Removing irrelevant/unnessasary words.
df2['filtered_token'] = df2['tokens'].apply(lambda tokens: [
                        word for word in tokens if word.lower() not in english_stopwords
                        and len(word) > 3
                        ])

df2['filtered_token'].head(4)
```

```
0    [jurong, point, crazy, available, bugis, great...
1                                              [joking]
2    [free, entry, wkly, comp, final, tkts, 21st, 2...
3                                    [early, already]
Name: filtered_token, dtype: object
```

The filtered tokens contain (or may contain) important words that differentiate spam and non-spam SMS messages from eachother. We will use these to identify the most important and frequent words that appear in spam SMS messages.

Given that our Neural Network model performed very well, when we extract the most important words within our SMS spam messages, we will utilize our predictions column (in df2) to extract all of our predicted spam messages. We will then use these specific messages to analyze our filtered tokens.

SMS Messages - Most Frequent Spam Words

```
Only_Spam_Messages = df2.loc[df2["predictions"] == 1]
Only_Spam_Messages.head(4)
```

	label	messages	predictions	tokens
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...
8	1	WINNER!! As a valued network customer you have...	1	[winner, as, a, valued, network, customer, you...
9	1	Had your mobile 11 months or more? U R entitle...	1	[had, your, mobile, 11, months, or, more, u, r...
11	1	SIX chances to win CASH! From 100 to 20,000 po...	1	[six, chances, to, win, cash, from, 100, to, 2...

```
Only_Spam_Messages_Updated = Only_Spam_Messages.explode('filtered_token')
Only_Spam_Messages_Updated
```

	label	messages	predictions	tokens	filter
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...	free
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...	entry
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...	wkly
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...	comp
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	1	[free, entry, in, 2, a, wkly, comp, to, win, f...	final
...	...	...	...	...	...
5567	1	This is the 2nd time we have tried 2 contact u...	1	[this, is, the, 2nd, time, we, have, tried, 2,...	0871
5567	1	This is the 2nd time we have tried 2 contact u...	1	[this, is, the, 2nd, time, we, have, tried, 2,...	now
5567	1	This is the 2nd time we have tried 2 contact u...	1	[this, is, the, 2nd, time, we, have, tried, 2,...	minu
5567	1	This is the 2nd time we have tried 2 contact u...	1	[this, is, the, 2nd, time, we, have, tried, 2,...	natio
5567	1	This is the 2nd time we have tried 2 contact u...	1	[this, is, the, 2nd, time, we, have, tried, 2,...	rate

```
TopWords = Only_Spam_Messages_Updated["filtered_token"].value_counts().head(15)
```

```
Spam_Words_SMS = TopWords.index.tolist()
```

Above is a list with the most frequent words use in SMS Messages that happen to be Spam Messages. Earlier, we made a list like this for the words that appear in Spam Emails as well.

Part 3: Top Spam Words

In this final part, we will construct a word cloud with the most frequent words that appear in both Spam SMS texts and Spam Emails. Below are the two lists that we created earlier that include the most important and frequent words that appear in Spam Messages. We will combine them to create a larger list including all the words.

```
# Spam_Words_Emails
```

```
# Spam_Words_SMS
```

```
Spam_Words = Spam_Words_Emails + Spam_Words_SMS
```

Lets drop any duplicates within the list since we do not want the same word appearing multiple times in a word cloud. We will use a dictionary since dictionaries do not contain any duplicate keys.

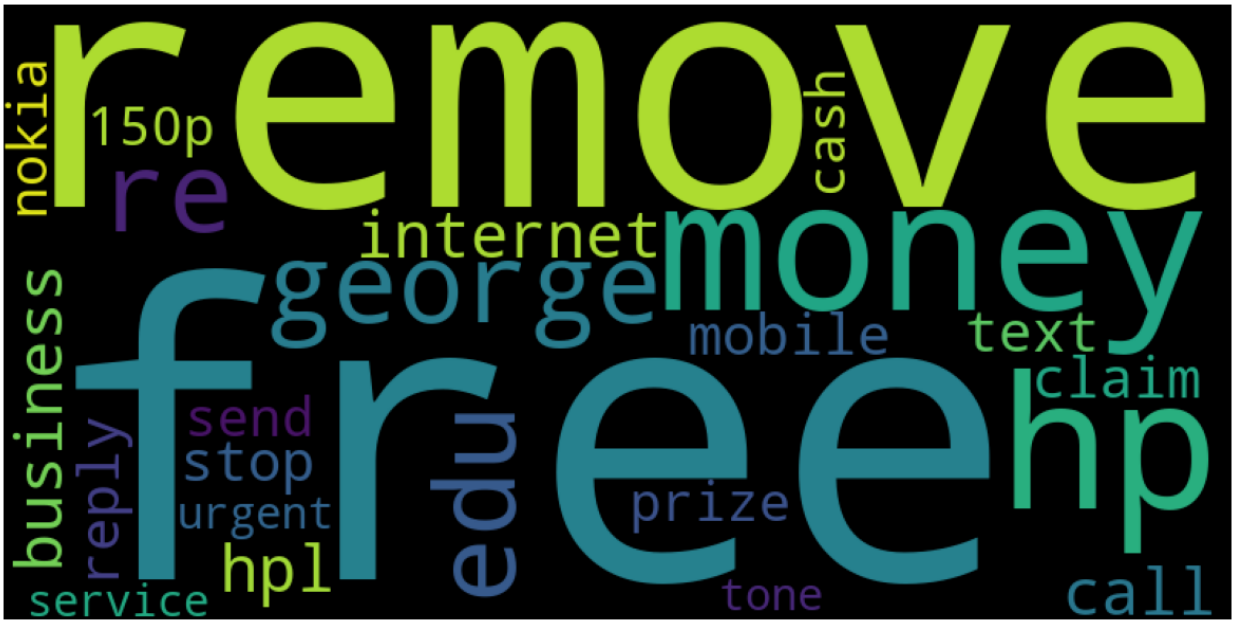
```
Spam_Words = list(dict.fromkeys(Spam_Words))
```

```
Spam_Words
```

```
['free',
 'remove',
 'your',
 'money',
 'hp',
 'our',
 '000',
 'you',
 'george',
 'edu',
 're',
 'internet',
 'business',
 'all',
 'hpl',
 'call',
 'mobile',
 'text',
 'stop',
 'claim',
 'reply',
 'prize',
 'cash',
```

```
 '150p',
 'nokia',
 'send',
 'urgent',
 'tone',
 'service']
```

```
WordCloud_Strings = (" ").join(Spam_Words)
wc = WordCloud(width = 800, height = 400, background_color="black")
wc.generate(WordCloud_Strings)

plt.figure(figsize=(15,8))
plt.imshow(wc)
plt.axis('off')
plt.show()
```



The word cloud above highlights the most frequent and most likely words the would come up in any sort of Spam message, whether it is an Email or and SMS Message. Individuals should avoid messages and/or emails that frequently contain the words above, or if they contain a large amount of the words in the cloud above.

Conclusion

So, we successfuly managed to use a Random Forest classifier to identify the most significant words in predicting whether or not an Email is Spam. We also successfuly used a Neural Network (single layer and 300 neurons) to predict whether or not SMS messages are spam. Because our Neural Network turned out to work extremely well, we

used our predicted SMS spam messages and identified the most frequent words within them, using simple techniques within Natural Language Processing. Finally, for the word cloud, majority of the results were well expected (for example, words like “Free”, “Cash”, “Prize”, “Money” and “Claim”). One word in the word cloud that came out as surprising was “George”. However, almost all of the other words were very well expected.