```
#  Spam Filter with Naive Bayes
#======================================

# Import libraries
import numpy as np
import pandas as pd
import re
import nltk
from IPython.display import display
from wordcloud import WordCloud, ImageColorGenerator
import matplotlib.pyplot as plt
from PIL import Image

# Activate the necessary magics
%matplotlib inline
%config InlineBackend.figure_format ='retina'

# Setting display options
pd.set_option('display.max_columns', None)  # or 1000
pd.set_option('display.max_rows', None)  # or 1000
pd.set_option('display.max_colwidth', -1)
# Read in data
spam_collection = pd.read_csv('SMSSpamCollection.csv', sep='\t', header=None,
names=['Label', 'SMS'])
print(spam_collection.shape)
spam_collection.head()

spam_collection.info()
spam_collection['Label'].value_counts(normalize = True)

# Original Data (13.4% of the messages are spam, while the rest are ham)

fig1, ax1 = plt.subplots(figsize=(5,5))

labels = ['Spam', 'Ham']
sizes = [len(spam_collection[spam_collection['Label'] == 'spam']),
len(spam_collection[spam_collection['Label'] == 'ham'])]
explode = (0, 0.1)  # only "explode" the 2nd slice

ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,
startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax1.set_title("Original Data Set", fontsize=14)

plt.show()

# Random data show

texts = spam_collection['SMS'].sum()
```

```
wc = WordCloud(max_words=1000,contour_width=3, contour_color='red')

wc.generate(texts)

plt.figure(figsize=[15,7])
plt.imshow(wc, interpolation='bilinear')
plt.axis("off")
plt.show()
plt.rcParams['savefig.dpi'] = 1100

#-------------------------------------------------
# Splitting Data
#A split-up of 80% and 20%, respectively

# Randomize the entire data set
randomized_collection = spam_collection.sample(frac=1, random_state=3)

# Calculate index for split
training_test_index = round(len(randomized_collection) * 0.8)

# Training/Test split
training_set = randomized_collection[:training_test_index].reset_index(drop=True)
test_set = randomized_collection[training_test_index:].reset_index(drop=True)
print('Training Data:')
print(training_set.shape)
print('Testing Data:')
print(test_set.shape)

print('Training set:\n', training_set['Label'].value_counts(normalize =
True),'\n\nTest set:')
test_set['Label'].value_counts(normalize = True)

# Training Set
fig2, ax2 = plt.subplots(figsize=(5,5))

labels = ['Spam', 'Ham']
sizes = [len(training_set[training_set['Label'] == 'spam']),
len(training_set[training_set['Label'] == 'ham'])]
explode = (0, 0.1)  # only "explode" the 2nd slice

ax2.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,
startangle=90)
ax2.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax2.set_title("Training Set", fontsize=14)

plt.show()

# Test Set

fig3, ax3 = plt.subplots(figsize=(5,5))
```

```
labels = ['Spam', 'Ham']
sizes = [len(test_set[test_set['Label'] == 'spam']), len(test_set[test_set['Label']
== 'ham'])]
explode = (0, 0.1)  # only "explode" the 2nd slice

ax3.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%', shadow=False,
startangle=90)
ax3.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.
ax3.set_title("Test Set", fontsize=14)

plt.show()

#-------------------------------------------------------
# MODEL DEVELOPMENT
#...............................

# Data Pre-Processing
#*************************************************
# 1. Normalization

# Original training set - before processing
training_set.head()

# Replace addresses (hhtp, email), numbers (plain, phone), money symbols
training_set['SMS'] =
training_set['SMS'].str.replace(r'\b[\w\-.]+?@\w+?\.\w{2,4}\b',
                                          ' ')
training_set['SMS'] =
training_set['SMS'].str.replace(r'(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)',
                                          ' ')
training_set['SMS'] = training_set['SMS'].str.replace(r'£|\$', ' ')
training_set['SMS'] =
training_set['SMS'].str.replace(r'\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s
.-]?\d{4}\b',
                                          ' ')
training_set['SMS'] = training_set['SMS'].str.replace(r'\d+(\.\d+)?', ' ')

# Remove punctuation, collapse all whitespace (spaces, line breaks, tabs) into a
single space & eliminate any leading/trailing whitespace.
training_set['SMS'] = training_set['SMS'].str.replace(r'[^\w\d\s]', ' ')
training_set['SMS'] = training_set['SMS'].str.replace(r'\s+', ' ')
training_set['SMS'] = training_set['SMS'].str.replace(r'^\s+|\s+?$', '')

# Lowercase the entire corpus
training_set['SMS'] = training_set['SMS'].str.lower()

training_set.head()
#--------------------------------
# Natural Language Tool
```

```
import nltk
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')
#.............................
# removing stopword like verb, article ect.
from nltk.corpus import stopwords
stop_words = nltk.corpus.stopwords.words('english')
training_set['SMS'] = training_set['SMS'].apply(lambda x: ' '.join(
    term for term in x.split() if term not in set(stop_words))
)
training_set.head()

# Lemmatization: the process of grouping together different inflected forms of the
same word
#.........................................................
lemmatizer = nltk.stem.WordNetLemmatizer()
training_set['SMS'] = training_set['SMS'].apply(lambda x: ' '.join(
    lemmatizer.lemmatize(term, pos='v') for term in x.split())
)
training_set.head()

#Stemming:reducing a word to its stem that affixes to suffixes and prefixes or to
the roots of words known as "lemmas".
#.................................................

porter = nltk.PorterStemmer()
training_set['SMS'] = training_set['SMS'].apply(lambda x: ' '.join(
    porter.stem(term) for term in x.split())
)

training_set.head()

# 2. Tokenization:  the process of replacing sensitive data with unique
identification symbols that retain all the essential information about the data
without compromising its security.
#**************************
training_set['SMS'] = training_set['SMS'].apply(lambda sms:
nltk.word_tokenize(sms))
training_set.head()

#-------------------------------
#Feature Extraction
#-------------------------------

corpus = training_set['SMS'].sum()
len(corpus)

# Transform the list to a set, to remove duplicates
```

```
temp_set = set(corpus)

# Revert to a list
vocabulary = list(temp_set)
len(vocabulary)

# Create the dictionary
len_training_set = len(training_set['SMS'])
word_counts_per_sms = {unique_word: [0] * len_training_set for unique_word in
vocabulary}

for index, sms in enumerate(training_set['SMS']):
    for word in sms:
        word_counts_per_sms[word][index] += 1
# Convert to dataframe
word_counts = pd.DataFrame(word_counts_per_sms)
word_counts.head()

word_counts.shape

# Concatenate with the original training set
training_set_final = pd.concat([training_set, word_counts], axis=1)
training_set_final.head()

#----------------------
# Calculating Probability
#---------------------------

# Filter the spam and ham dataframes
spam_df = training_set_final[training_set_final['Label'] == 'spam'].copy()
ham_df = training_set_final[training_set_final['Label'] == 'ham'].copy()
# Calculate P(Spam) and P(Ham)
p_spam = spam_df.shape[0] / training_set_final.shape[0]
p_ham = ham_df.shape[0] / training_set_final.shape[0]
print(p_spam)
p_ham

# Calculate Nspam, Nham and Nvocabulary
spam_words_per_message = spam_df['SMS'].apply(len)
n_spam = spam_words_per_message.sum()

ham_words_per_message = ham_df['SMS'].apply(len)
n_ham = ham_words_per_message.sum()

n_vocabulary = len(vocabulary)
# Opting for the Laplace smoothing to remove 0 probability problem
alpha = 1

#-----------
#Calculating Parameters
```

```
# P(wi|Spam) and P(wi|Ham) depend on the training set, which doesn't change, thus
they are constant.

# Create two dictionaries that match each unique word with the respective
probability value.
parameters_spam = {unique_word: 0 for unique_word in vocabulary}
parameters_ham = {unique_word: 0 for unique_word in vocabulary}

# Iterate over the vocabulary and for each word, calculate P(wi|Spam) and P(wi|Ham)
for unique_word in vocabulary:
    p_unique_word_spam = (spam_df[unique_word].sum() + alpha) / (n_spam + alpha *
n_vocabulary)
    p_unique_word_ham = (ham_df[unique_word].sum() + alpha) / (n_ham + alpha *
n_vocabulary)

    # Update the calculated propabilities to the dictionaries
    parameters_spam[unique_word] = p_unique_word_spam
    parameters_ham[unique_word] = p_unique_word_ham
#-------------------------------------------------------
#Classifying A New Message
#.................................

def sms_classify(message):
    '''
    Takes in as input a new sms (w1, w2, ..., wn),
    calculates P(Spam|w1, w2, ..., wn) and P(Ham|w1, w2, ..., wn),
    compares them and outcomes whether the message is spam or not.
    '''

    # Replace addresses (hhtp, email), numbers (plain, phone), money symbols
    message = message.replace(r'\b[\w\-.]+?@\w+?\.\w{2,4}\b', ' ')
    message = message.replace(r'(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)', ' ')
    message = message.replace(r'£|\$', ' ')
    message =
message.replace(r'\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}\b', '
')
    message = message.replace(r'\d+(\.\d+)?', ' ')

    # Remove punctuation, collapse all whitespace (spaces, line breaks, tabs) into
a single space & eliminate any leading/trailing whitespace.
    message = message.replace(r'[^\w\d\s]', ' ')
    message = message.replace(r'\s+', ' ')
    message = message.replace(r'^\s+|\s+?$', '')

    # Lowercase the entire corpus
    message = message.lower()

    # Remove stop words
    terms = []
    for term in message.split():
```

```
        if term not in set(stop_words):
            terms.append(term)
            message = ' '.join(terms)

    # Lemmatization
    message = ' '.join(lemmatizer.lemmatize(term, pos='v') for term in
message.split())

    # Stemming
    message = ' '.join(porter.stem(term) for term in message.split())

    # Tokenization
    message = message.split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    print('P(Spam|message):', p_spam_given_message)
    print('P(Ham|message):', p_ham_given_message)

    if p_ham_given_message > p_spam_given_message:
        print('Label: Ham')
    elif p_ham_given_message < p_spam_given_message:
        print('Label: Spam')
    else:
        print('Equal probabilities ~ Human action needed!')

print("Test with message: Hey, Sign up with this promo code and get your card for
amazing exchange fees abroad and £5 to spend anywhere! Promocode: D48KV7BN")

sms_classify('''Hey, Sign up with this promo code and get your card for amazing
                exchange fees abroad and £5 to spend anywhere! Promocode:
D48KV7BN''')

print('Test with message: Okey Stan! Seems to be a reasonable amount of money. Ill
think of it and let you know ASAP.')

sms_classify('''Okey Stan! Seems to be a reasonable amount of money. I'll think of
it and let you know ASAP.''')

print('Test with any massage')
sms_classify(input('input text:'))
```

```
#--------------------------------------
#Measuring the Spam Filter's Accuracy

# We define the classify () function again, this time returning the outcomes
def sms_classify_test_set(message):
    '''
    Takes in as input a new sms (w1, w2, ..., wn),
    calculates P(Spam|w1, w2, ..., wn) and P(Ham|w1, w2, ..., wn),
    compares them and returns the spam or ham label, respectively.
    '''

    # Replace addresses (hhtp, email), numbers (plain, phone), money symbols
    message = message.replace(r'\b[\w\-.]+?@\w+?\.\w{2,4}\b', ' ')
    message = message.replace(r'(http[s]?\S+)|(\w+\.[A-Za-z]{2,4}\S*)', ' ')
    message = message.replace(r'£|\$', ' ')
    message =
message.replace(r'\b(\+\d{1,2}\s)?\d?[\-(.]?\d{3}\)?[\s.-]?\d{3}[\s.-]?\d{4}\b', '
')
    message = message.replace(r'\d+(\.\d+)?', ' ')

    # Remove punctuation, collapse all whitespace (spaces, line breaks, tabs) into
a single space & eliminate any leading/trailing whitespace.
    message = message.replace(r'[^\w\d\s]', ' ')
    message = message.replace(r'\s+', ' ')
    message = message.replace(r'^\s+|\s+?$', '')

    # Lowercase the entire corpus
    message = message.lower()

    # Remove stop words
    terms = []
    for term in message.split():
        if term not in set(stop_words):
            terms.append(term)
            message = ' '.join(terms)

    # Lemmatization
    message = ' '.join(lemmatizer.lemmatize(term, pos='v') for term in
message.split())

    # Stemming
    message = ' '.join(porter.stem(term) for term in message.split())

    # Tokenization
    message = message.split()

    p_spam_given_message = p_spam
    p_ham_given_message = p_ham

    for word in message:
```

```
        if word in parameters_spam:
            p_spam_given_message *= parameters_spam[word]

        if word in parameters_ham:
            p_ham_given_message *= parameters_ham[word]

    if p_ham_given_message > p_spam_given_message:
        return 'ham'
    elif p_spam_given_message > p_ham_given_message:
        return 'spam'
    else:
        return 'needs human classification'
test_set['sms_predicted'] = test_set['SMS'].apply(sms_classify_test_set)
test_set.head()

# Calculate the accuracy of the algorithm
correct = 0
total = test_set.shape[0]

for row in test_set.iterrows():
    row = row[1]
    if row['Label'] == row['sms_predicted']:
        correct += 1

print('Results \n-------')
print('Valid:', correct)
print('Invalid:', total - correct)
print('Accuracy:', round(correct/total, 4))
```