Introduction

The upsurge in the volume of unwanted emails called spam has created an intense need for the development of more dependable and robust antispam filters. Any promotional messages or advertisements that end up in our inbox can be categorised as spam as they don't provide any value and often irritates us.

Overview of the Dataset used

We will make use of the SMS spam classification data.

> *The SMS Spam Collection is a set of SMS tagged messages that have been collected for SMS Spam research. It contains one set of SMS messages in English of 5,574 messages, tagged according to being ham (legitimate) or spam.*

The data was obtained from UCI's Machine Learning Repository, alternatively, I have also uploaded the dataset and completed Jupiter notebook onto my GitHub repo.

In this article, we'll discuss:

Data processing

- Import the required packages
- Loading the Dataset
- Remove the unwanted data columns
- Preprocessing and Exploring the Dataset
- Build word cloud to see which message is spam and which is not.
- Remove the stop words and punctuations
- Convert the text data into vectors

Building a sms spam classification model

- Split the data into train and test sets
- Use Sklearn built-in classifiers to build the models
- Train the data on the model
- Make predictions on new data

Import the required packages

```
%matplotlib inline
import matplotlib.pyplot as plt
import csv
import sklearn
import pickle
from wordcloud import WordCloud
import pandas as pd
import numpy as np
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import CountVectorizer, TfidfTransformer
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV,train_test_split,StratifiedKFo
```

please note! You might find that I have reimported some of these packages again later in the article, it is just for ease of use if I ever have to use those code blocks again in future projects, you may omit those.😉

Loading the Dataset

```
data = pd.read_csv('dataset/spam.csv', encoding='latin-1')
data.head()
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	Go until jurong point, crazy.. Available only ...	NaN	NaN	NaN
1	ham	Ok lar... Joking wif u oni...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I don't think he goes to usf, he lives aro...	NaN	NaN	NaN

Removing unwanted columns

From the above figure, we can see that there are some unnamed columns and the label and text column name is not intuitive so let's fix those in this step.

```
data = data.drop(["Unnamed: 2", "Unnamed: 3", "Unnamed: 4"], axis=1)
data = data.rename(columns={"v2" : "text", "v1":"label"})
data[1990:2000]
```

	label	text
1990	ham	HI DARLIN IVE JUST GOT BACK AND I HAD A REALLY...
1991	ham	No other Valentines huh? The proof is on your ...
1992	spam	Free tones Hope you enjoyed your new content. ...
1993	ham	Eh den sat u book e kb liao huh...
1994	ham	Have you been practising your curtsey?
1995	ham	Shall i come to get pickle
1996	ham	Lol boo I was hoping for a laugh
1997	ham	\YEH I AM DEF UP4 SOMETHING SAT
1998	ham	Well, I have to leave for my class babe ... Yo...
1999	ham	LMAO where's your fish memory when I need it?

now that the data is looking pretty, let's move on.

```
data['label'].value_counts()

# OUTPUT
ham     4825
spam     747
Name: label, dtype: int64
```

Preprocessing and Exploring the Dataset

If you are completely new to NLTK and Natural Language Processing(NLP) I would recommend checking out this short article before continuing. Introduction to Word Frequencies in NLP

```
# Import nltk packages and Punkt Tokenizer Models
import nltk
nltk.download("punkt")
import warnings
warnings.filterwarnings('ignore')
```

Build word cloud to see which message is spam and which is

Build word cloud to see which message is spam and which is not

> *ham words are the opposite of spam in this dataset, 🤷 yeah I also don't have any clue why it is so.*

```
ham_words = ''
spam_words = ''
```

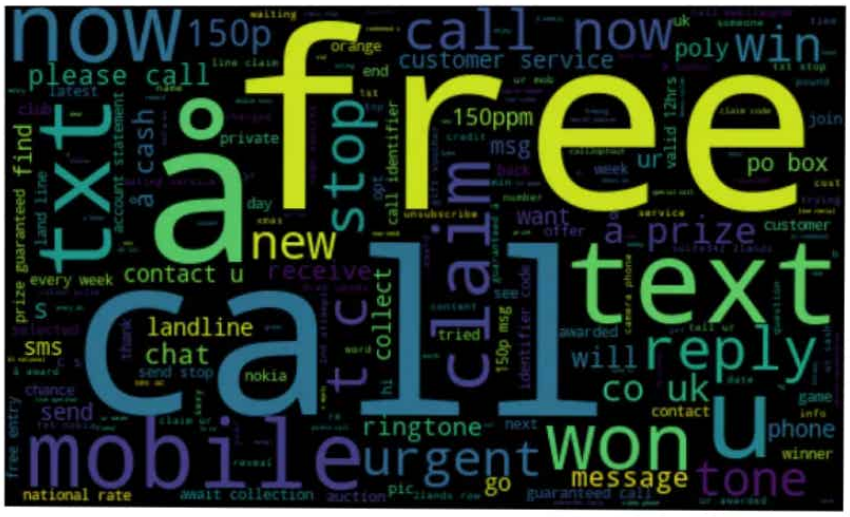
```
# Creating a corpus of spam messages
for val in data[data['label'] == 'spam'].text:
    text = val.lower()
    tokens = nltk.word_tokenize(text)
    for words in tokens:
        spam_words = spam_words + words + ' '

# Creating a corpus of ham messages
for val in data[data['label'] == 'ham'].text:
    text = text.lower()
    tokens = nltk.word_tokenize(text)
    for words in tokens:
        ham_words = ham_words + words + ' '
```

let's use the above functions to create Spam word cloud and ham word cloud.

```
spam_wordcloud = WordCloud(width=500, height=300).generate(spam_words)
ham_wordcloud = WordCloud(width=500, height=300).generate(ham_words)
```

```
#Spam Word cloud
plt.figure( figsize=(10,8), facecolor='w')
plt.imshow(spam_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```




```
#Creating Ham wordcloud
plt.figure( figsize=(10,8), facecolor='g')
plt.imshow(ham_wordcloud)
plt.axis("off")
plt.tight_layout(pad=0)
plt.show()
```

from the spam word cloud, we can see that "free" is most often used in spam.

Now, we can convert the `spam` and `ham` into 0 and 1 respectively so that the machine can understand.

```
data = data.replace(['ham','spam'],[0, 1])
data.head(10)
```

	label	text
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...
4	0	Nah I don't think he goes to usf, he lives aro...
5	1	FreeMsg Hey there darling it's been 3 week's n...
6	0	Even my brother is not like to speak with me. ...
7	0	As per your request 'Melle Melle (Oru Minnamin...
8	1	WINNER!! As a valued network customer you have...
9	1	Had your mobile 11 months or more? U R entitle...

Removing punctuation and stopwords from the messages

Punctuation and stop words do not contribute anything to our model, so we have to remove them. Using NLTK library we can easily do it.

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

#remove the punctuations and stopwords
import string
def text_process(text):

    text = text.translate(str.maketrans('', '', string.punctuation))
    text = [word for word in text.split() if word.lower() not in stopwords.words

    return " ".join(text)

data['text'] = data['text'].apply(text_process)
data.head()
```

	label	text
0	0	Go jurong point crazy Available bugis n great ...
1	0	Ok lar Joking wif u oni
2	1	Free entry 2 wkly comp win FA Cup final tkts 2...
3	0	U dun say early hor U c already say
4	0	Nah dont think goes usf lives around though

Now, create a data frame from the processed data before moving to the next step.

```
text = pd.DataFrame(data['text'])
label = pd.DataFrame(data['label'])
```

Converting words to vectors using Count Vectorizer

```
## Counting how many times a word appears in the dataset

from collections import Counter

total_counts = Counter()
for i in range(len(text)):
    for word in text.values[i][0].split(" "):
        total_counts[word] += 1

print("Total words in data set: ", len(total_counts))

# OUTPUT
Total words in data set:  11305
```

```
# Sorting in decreasing order (Word with highest frequency appears first)
vocab = sorted(total_counts, key=total_counts.get, reverse=True)
print(vocab[:60])

# OUTPUT
['u', '2', 'call', 'U', 'get', 'Im', 'ur', '4', 'ltgt', 'know', 'go', 'like', '
```

```
# Mapping from words to index

vocab_size = len(vocab)
word2idx = {}
#print vocab_size
for i, word in enumerate(vocab):
    word2idx[word] = I
```

```
# Text to Vector
def text_to_vector(text):
    word_vector = np.zeros(vocab_size)
    for word in text.split(" "):
        if word2idx.get(word) is None:
            continue
        else:
            word_vector[word2idx.get(word)] += 1
    return np.array(word_vector)
```

```
# Convert all titles to vectors
word_vectors = np.zeros((len(text), len(vocab)), dtype=np.int_)
for i, (_, text_) in enumerate(text.iterrows()):
    word_vectors[i] = text_to_vector(text_[0])

word_vectors.shape

# OUTPUT
(5572, 11305)
```

Converting words to vectors using TF-IDF Vectorizer

```
#convert the text data into vectors
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer()
vectors = vectorizer.fit_transform(data['text'])
vectors.shape

# OUTPUT
(5572, 9376)
```

Splitting into training and test set

```
#split the dataset into train and test set
X_train, X_test, y_train, y_test = train_test_split(features, data['label'], te
```

Classifying using sklearn's pre-built classifiers

In this step we will use some of the most popular classifiers out there and compare their results.

Classifiers used:

1. spam classifier using logistic regression
2. email spam classification using Support Vector Machine(SVM)
3. spam classifier using naive bayes
4. spam classifier using decision tree
5. spam classifier using K-Nearest Neighbor(KNN)
6. spam classifier using Random Forest Classifier

We will make use of sklearn library. This amazing library has all of the above algorithms we just have to import them and it is as easy as that. No need to worry about all the maths and statistics behind it.

```
#import sklearn packages for building classifiers
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

```
#initialize multiple classification models
svc = SVC(kernel='sigmoid', gamma=1.0)
knc = KNeighborsClassifier(n_neighbors=49)
mnb = MultinomialNB(alpha=0.2)
dtc = DecisionTreeClassifier(min_samples_split=7, random_state=111)
lrc = LogisticRegression(solver='liblinear', penalty='l1')
rfc = RandomForestClassifier(n_estimators=31, random_state=111)
```

```
#create a dictionary of variables and models
clfs = {'SVC' : svc,'KN' : knc, 'NB': mnb, 'DT': dtc, 'LR': lrc, 'RF': rfc}
```

```
#fit the data onto the models
def train(clf, features, targets):
    clf.fit(features, targets)

def predict(clf, features):
    return (clf.predict(features))
```

```
pred_scores_word_vectors = []
for k,v in clfs.items():
    train(v, X_train, y_train)
    pred = predict(v, X_test)
    pred_scores_word_vectors.append((k, [accuracy_score(y_test , pred)]))
```

Predictions using TFIDF Vectorizer algorithm

```
pred_scores_word_vectors

# OUTPUT
[('SVC', [0.9784688995215312]),
 ('KN', [0.9330143540669856]),
 ('NB', [0.9880382775119617]),
 ('DT', [0.9605263157894737]),
 ('LR', [0.9533492822966507]),
 ('RF', [0.9796650717703349])]
```

Model predictions

```
#write functions to detect if the message is spam or not
def find(x):
    if x == 1:
        print ("Message is SPAM")
    else:
        print ("Message is NOT Spam")
```

```
newtext = ["Free entry"]
integers = vectorizer.transform(newtext)
```

```
x = mnb.predict(integers)
find(x)

# OUTPUT
Message is SPAM
```

Checking Classification Results with Confusion Matrix

> *If you are confused about the confusion matrix, read this small article before proceeding - The ultimate guide to confusion matrix in machine learning*

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
# Naive Bayes
y_pred_nb = mnb.predict(X_test)
y_true_nb = y_test
cm = confusion_matrix(y_true_nb, y_pred_nb)
f, ax = plt.subplots(figsize =(5,5))
sns.heatmap(cm,annot = True,linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)
plt.xlabel("y_pred_nb")
plt.ylabel("y_true_nb")
plt.show()
```