SPAM Detection Modelling using ML

Python Libraries Installations

In [1]:

```
!pip install plotly
!pip install lazypredict
!pip install scikit-learn
```

```
Requirement already satisfied: plotly in c:\programdata\anaconda3\lib\site-packages (4.8.2)
Requirement already satisfied: retrying>=1.3.3 in c:\programdata\anaconda3\lib\site-packages (from pl
otly) (1.3.3)
Requirement already satisfied: six in c:\programdata\anaconda3\lib\site-packages (from plotly) (1.12.
0)
Requirement already satisfied: lazypredict in c:\programdata\anaconda3\lib\site-packages (0.2.7)
Requirement already satisfied: Click>=7.0 in c:\programdata\anaconda3\lib\site-packages (from lazypre
dict) (7.0)
Requirement already satisfied: scikit-learn in c:\programdata\anaconda3\lib\site-packages (0.22)
Requirement already satisfied: joblib>=0.11 in c:\programdata\anaconda3\lib\site-packages (from sciki
t-learn) (0.14.1)
Requirement already satisfied: numpy>=1.11.0 in c:\programdata\anaconda3\lib\site-packages (from scik
it-learn) (1.16.6)
Requirement already satisfied: scipy>=0.17.0 in c:\programdata\anaconda3\lib\site-packages (from scik
it-learn) (1.4.1)
```

Importing libraries

In [2]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import plotly.express as px

from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import *
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.pipeline import Pipeline
from sklearn import metrics

from lazypredict.Supervised import LazyClassifier # for evaluating multiple models
```

```
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\utils\deprecation.py:144: FutureWarning: The sklea
rn.utils.testing module is  deprecated in version 0.22 and will be removed in version 0.24. The corre
sponding classes / functions should instead be imported from sklearn.utils. Anything that cannot be i
mported from sklearn.utils is now part of the private API.
  warnings.warn(message, FutureWarning)
```

Data Loading and Exploration

In [3]:

```
df = pd.read_csv('spam.csv',encoding='ISO-8859-1')
```

In [4]:

```
df.head()
```

Out[4]:

	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

In [5]:

```
df.columns
```

Out[5]:

```
Index(['v1', 'v2', 'Unnamed: 2', 'Unnamed: 3', 'Unnamed: 4'], dtype='object')
```

We can see that there are there unnecessary columns in the data set.We can drop the unwanted columns

In [6]:

```python
df.drop(columns=['Unnamed: 2', 'Unnamed: 3','Unnamed: 4'],inplace=True)
```

In [7]:

```python
df.head()
```

Out[7]:

	v1	v2
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...
4	ham	Nah I don't think he goes to usf, he lives aro...

Lets change the names of columns for convienence

In [8]:

```python
df.rename({'v1': 'label', 'v2': 'messages'}, axis=1, inplace=True)
```

In [9]:

```python
df.sample()
```

Out[9]:

	label	messages
926	ham	But I'm on a diet. And I ate 1 too many slices...

In [10]:

```python
df.describe()
```

Out[10]:

	label	messages
count	5572	5572
unique	2	5169
top	ham	Sorry, I'll call later
freq	4825	30

In [11]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
label       5572 non-null object
messages    5572 non-null object
dtypes: object(2)
memory usage: 87.2+ KB
```

In [12]:

```python
df.groupby('label').describe().transpose()
```

Out[12]:

	label	ham	spam
messages	count	4825	747
	unique	4516	653
	top	Sorry, I'll call later	Please call our customer service representativ...
	freq	30	4

In [13]:

```python
df.isnull().sum()
```

Out[13]:

```
label       0
messages    0
dtype: int64
```

In [14]:

```python
df.shape
```

Out[14]:

```
(5572, 2)
```

Create a new column of length of message

In [15]:

```python
df['length']=df['messages'].apply(len)
```

In [16]:

```python
df.head()
```

Out[16]:

	label	messages	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

In [17]:

```
df['label'].unique()
```

Out[17]:

```
array(['ham', 'spam'], dtype=object)
```

Get Target data information

In [18]:

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

Out[18]:

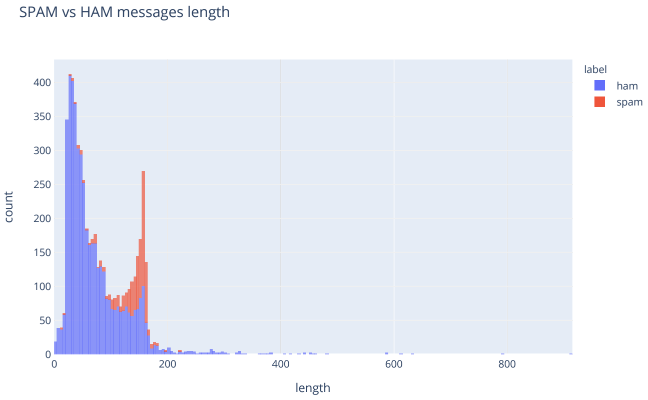
```
ham     4825
spam     747
Name: label, dtype: int64
```

Target is imbalanced, we have to be careful while choosing the evaluation metric.

Plotting the histogram of Labels

In [19]:

```
px.histogram(data_frame=df,x='length',color='label',opacity=.7,title='SPAM vs HAM messages length')
```

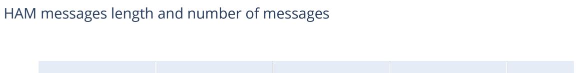


SPAM vs HAM messages length

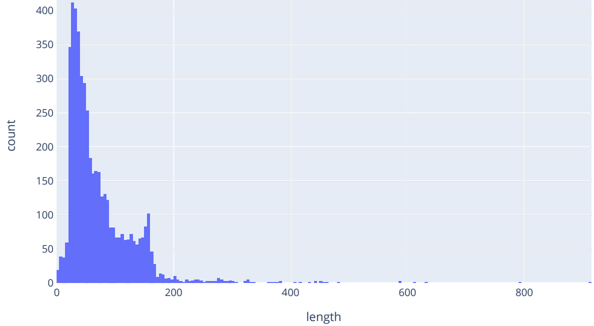
Spam text messages are longer than ham text messages

In [20]:

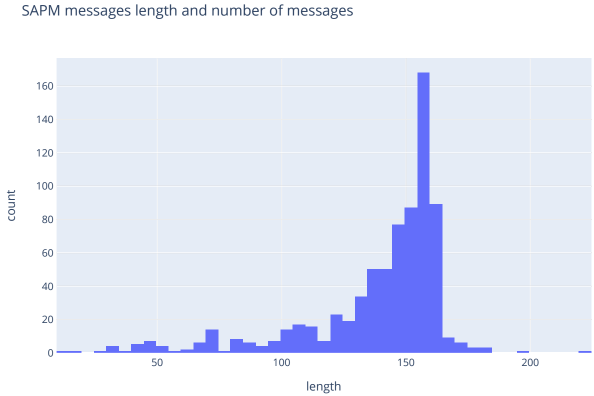
```
px.histogram(data_frame=df[df['label']=='ham'],x='length',title='HAM messages length and number of mess
ages')
```



HAM messages length and number of messages



```
In [22]: px.histogram(data_frame=df[df['label']=='spam'],x='length',title='SAPM messages length and number of messages')
```


SAPM messages length and number of messages

Lets print out longest message

```
In [23]: df['length'].describe()
```

```
Out[23]: count    5572.00
         mean       80.12
         std        59.69
         min         2.00
         25%        36.00
         50%        61.00
         75%       121.00
         max       910.00
         Name: length, dtype: float64
```

```
In [24]: df[df['length']==910]['messages'].iloc[0]
```

Out[24]:

```
"For me the love should start with attraction.i should feel that I need her every time around me.she
should be the first thing which comes in my thoughts.I would start the day and end it with her.she sh
ould be there every time I dream.love will be then when my every breath has her name.my life should h
appen around her.my life will be named to her.I would cry for her.will give all my happiness and take
all her sorrows.I will be ready to fight with anyone for her.I will be in love when I will be doing t
he craziest things for her.love will be when I don't have to proove anyone that my girl is the most b
eautiful lady on the whole planet.I will always be singing praises for her.love will be when I start
up making chicken curry and end up makiing sambar.life will be the most beautiful then.will get every
morning and thank god for the day because she is with me.I would like to say a lot..will tell late
r.."
```

Insights:

The message length is important feature as we have observed from the above plots, as we see 2 different peaks for spam and ham messages.

Lets try a classifier only based on the length of message

In [25]:

```python
X = df['length'].values[:, None]
y = df['label']
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=2020)
```

LazyClassifier for evaluating multiple models

In [27]:

```python
from lazypredict.Supervised import LazyClassifier
clf = LazyClassifier(verbose=0,ignore_warnings=True, custom_metric=None)
models,predictions = clf.fit(X_train, X_test, y_train, y_test)
```

```
100%|██████████| 30/30 [00:09<00:00,  3.11it/s]
```

In [28]:

```python
pd.DataFrame(models)
```

Out[28]:

Model	Accuracy	Balanced Accuracy	ROC AUC	F1 Score	Time Taken
NearestCentroid	0.80	0.82	None	0.83	0.07
SVC	0.88	0.79	None	0.89	0.37
LabelPropagation	0.88	0.76	None	0.88	1.27
LabelSpreading	0.88	0.76	None	0.88	1.66
XGBClassifier	0.88	0.73	None	0.88	0.37
RandomForestClassifier	0.87	0.71	None	0.87	0.64
DecisionTreeClassifier	0.87	0.71	None	0.87	0.09
ExtraTreeClassifier	0.87	0.71	None	0.87	0.09
ExtraTreesClassifier	0.87	0.71	None	0.87	0.57
BaggingClassifier	0.88	0.70	None	0.87	0.13
LGBMClassifier	0.88	0.69	None	0.87	0.25
KNeighborsClassifier	0.86	0.69	None	0.86	0.23
AdaBoostClassifier	0.87	0.63	None	0.86	0.70
PassiveAggressiveClassifier	0.84	0.50	None	0.80	0.08
DummyClassifier	0.76	0.50	None	0.76	0.08
GaussianNB	0.87	0.50	None	0.80	0.11
Perceptron	0.87	0.50	None	0.80	0.08
QuadraticDiscriminantAnalysis	0.87	0.50	None	0.80	0.07
SGDClassifier	0.87	0.50	None	0.80	0.09
BernoulliNB	0.87	0.50	None	0.80	0.09
RidgeClassifier	0.85	0.49	None	0.80	0.09
RidgeClassifierCV	0.85	0.49	None	0.80	0.10
LogisticRegression	0.85	0.49	None	0.79	0.09

CalibratedClassifierCV	0.85	0.49	None	0.79	1.62
LinearSVC	0.85	0.49	None	0.80	0.45
LinearDiscriminantAnalysis	0.84	0.49	None	0.79	0.10

Insights:

The data is imbalanced so accuracy will not be a better metric to use, consider the scenario even if the model predicts all the data as `ham` the classifier would return an accuracy of 86.593% . Now the best thing in this scenario is to identify the business requirement, if we need to identify `spam` or `ham` and we can have precision or recall score. If both are important we can choose the `f1-score` which is harmonic mean of TPR and FPR.

Now considering `f1-score` , we can see that SVC classifier works well with the data, however we haven't considered the actual message as input. Lets try to build model based on the text message.

Making features from Text

In [29]:
```
df.head()
```

Out[29]:

	label	messages	length
0	ham	Go until jurong point, crazy.. Available only ...	111
1	ham	Ok lar... Joking wif u oni...	29
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	ham	U dun say so early hor... U c already then say...	49
4	ham	Nah I don't think he goes to usf, he lives aro...	61

In [30]:
```
X=df['messages']
```

In [31]:
```
y=df['label']
```

In [32]:
```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.33,
                                                    random_state=2020)
```

CountVectorizer:

`CountVectorizer` Converts a collection of text documents to a matrix of token counts

This implementation produces a sparse representation of the counts using scipy.sparse.csr_matrix.

In [33]:
```
count_vect=CountVectorizer()
```

In [34]:
```
X_train_counts=count_vect.fit_transform(X_train) # One step Fit and Transform
```

In [36]:
```
X_train.shape
```

Out[36]:
```
(3733,)
```

In [37]:
```
#Lets check the transformed matrix shape
X_train_counts.shape
```

Out[37]:
```
(3733, 7026)
```

TfidfTransformer

Transform a count matrix to a normalized tf or tf-idf representation

Tf means term-frequency while tf-idf means term-frequency times inverse document-frequency. This is a common term weighting scheme in information retrieval, that has also found good use in document classification.

The goal of using tf-idf instead of the raw frequencies of occurrence of a token in a given document is to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus.

In [38]:

```
tfidf_transformer=TfidfTransformer()
```

In [39]:

```
X_train_tfidf=tfidf_transformer.fit_transform(X_train_counts)
```

In [40]:

```
X_train_tfidf.shape
```

Out[40]:

```
(3733, 7026)
```

TfidfVectorizer

Convert a collection of raw documents to a matrix of TF-IDF features.

Equivalent to CountVectorizer followed by TfidfTransformer.

In [41]:

```
vectorizer=TfidfVectorizer()
```

In [42]:

```
X_train_tfidf=vectorizer.fit_transform(X_train)
```

SVC model

Linear SVC is better for text classification

If it is the simpler algorithm, why is the linear kernel recommended for text classification?

Text is often linearly separable, Most of text classification problems are linearly separable.

Text has a lot of features The linear kernel is good when there is a lot of features. That's because mapping the data to a higher dimensional space does not really improve the performance. In text classification, both the numbers of instances (document) and features (words) are large

Linear kernel is faster Training a SVM with a linear kernel is faster than with another kernel.

Conclusion:

Linear kernel is indeed very well suited for text-categorization.

Keep in mind however that it is not the only solution and in some case using another kernel might be better.

The recommended approach for text classification is to try a linear kernel first, because of its advantages. If however you search to get the best possible classification performance

In [43]:

```
clf=LinearSVC()
```

In [44]:

```
clf.fit(X_train_tfidf,y_train)
```

Out[44]:

```
LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
          intercept_scaling=1, loss='squared_hinge', max_iter=1000,
          multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
          verbose=0)
```

Pipleline with TfidfVectorizer and LinearSVC

In [45]:

```
text_clf=Pipeline([('tfidf',TfidfVectorizer()),('clf',LinearSVC())])
```

In [46]:

```
text_clf.fit(X_train,y_train)
```

Out[46]:

```
Pipeline(memory=None,
         steps=[('tfidf',
                 TfidfVectorizer(analyzer='word', binary=False,
                                 decode_error='strict',
                                 dtype=<class 'numpy.float64'>,
                                 encoding='utf-8', input='content',
                                 lowercase=True, max_df=1.0, max_features=None,
                                 min_df=1, ngram_range=(1, 1), norm='l2',
                                 preprocessor=None, smooth_idf=True,
                                 stop_words=None, strip_accents=None,
                                 sublinear_tf=False,
                                 token_pattern='(?u)\\b\\w\\w+\\b',
                                 tokenizer=None, use_idf=True,
                                 vocabulary=None)),
                ('clf',
```

```
                              LinearSVC(C=1.0, class_weight=None, dual=True,
                                        fit_intercept=True, intercept_scaling=1,
                                        loss='squared_hinge', max_iter=1000,
                                        multi_class='ovr', penalty='l2', random_state=None,
                                        tol=0.0001, verbose=0))],
                      verbose=False)
```

```
In [47]: predictions=text_clf.predict(X_test)
```

Model evaluation metrics

```
In [48]: print(confusion_matrix(y_test,predictions))
```

```
[[1584    4]
 [  27  224]]
```

we were able to classify almost all the ham messages except for 27 + 4 records

```
In [49]: print(classification_report(y_test,predictions))
```

```
              precision    recall  f1-score   support

         ham       0.98      1.00      0.99      1588
        spam       0.98      0.89      0.94       251

    accuracy                           0.98      1839
   macro avg       0.98      0.94      0.96      1839
weighted avg       0.98      0.98      0.98      1839
```

```
In [50]: metrics.accuracy_score(y_test,predictions)
Out[50]: 0.9831430125067971
```

```
In [51]: metrics.f1_score(y_test,predictions,pos_label="ham")
Out[51]: 0.9903094717099094
```

Insights:

We already have an fl_score close to 99% with defualt model and there is no need to tune further as we might run into `overfitting`

Lets try to predict this model on new data which is custom to verify its accuracy on the data which model has not seen.

Predicting on new data

```
In [52]: text_clf.predict(["you have won a lottery of $400000"])
Out[52]: array(['spam'], dtype=object)
```

```
In [53]: text_clf.predict(["honey i would be  late for dinner"])
Out[53]: array(['ham'], dtype=object)
```

```
In [54]: text_clf.predict(["your account has been debited 40,000"])
Out[54]: array(['spam'], dtype=object)
```

```
In [55]: text_clf.predict(["please stop going out to stop the spread of covid"])
Out[55]: array(['spam'], dtype=object)
```

```
In [56]: text_clf.predict(["remember our neighbour in blr, he has got covid"])
Out[56]: array(['ham'], dtype=object)
```

```
In [57]: text_clf.predict(["hey,........ hw u dng"])
Out[57]: array(['ham'], dtype=object)
```

```
In [58]: text_clf.predict([":) :) :P"])
Out[58]: array(['ham'], dtype=object)
```

Final Thoughts:

1. The model is able to idnetify all the ham and spam messages correctly
2. The model is even able to work on emojis in messages
3. LinearSVC model chosen for the task is better as its simple and training is faster
4. Features generated from `TfidfVectorizer` is better than the normal model with length of message as feature.
5. For imbalanced data problem, we should use alternate methods than the basic accuracy score as it won't tell the full story.