Data Prep for Kaggle Spam Data - EDA, Data Cleansing, Text Pre-Processing, and Tokenization

Kaggle Database Link

https://www.kaggle.com/datasets/uciml/sms-spam-collection-dataset

Load Libraries

In [1]:
```
import pandas as pd
import numpy as np
import itertools
import collections
import re
import nltk
import string
import opendatasets as od
import pickle
from nltk.corpus import stopwords
from nltk import bigrams
from nltk.stem.porter import PorterStemmer
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn import metrics, svm
from sklearn.metrics import precision_score, recall_score, roc_curve, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from keras.layers import SimpleRNN, LSTM, Dense, Dropout, Activation, Flatten
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, Extra
from xgboost import XGBClassifier, XGBRFClassifier
from sklearn.model_selection import RandomizedSearchCV
from imblearn.under_sampling import RandomUnderSampler
from imblearn.pipeline import Pipeline
from hyperopt import STATUS_OK, Trials, fmin, hp, tpe
```

Importing Data

In [2]:
```
#Loading corpus into data frame
df = pd.read_csv("Data/spam.csv", encoding = "ISO-8859-1", engine = "python")
print(df.shape)
```
```
(5572, 5)
```

In [3]:
```
df.head()
```

Out[3]:

	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

Cleaning Data

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   v1          5572 non-null   object
 1   v2          5572 non-null   object
 2   Unnamed: 2  50 non-null     object
 3   Unnamed: 3  12 non-null     object
 4   Unnamed: 4  6 non-null      object
dtypes: object(5)
memory usage: 217.8+ KB
```

In [5]:
```
#looking at the data in the unnamed columns
df[df['Unnamed: 2'].isnull() == False].head()
```

Out[5]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
95	spam	Your free ringtone is waiting to be collected....	PO Box 5249	MK17 92H. 450Ppw 16"	NaN
281	ham	\Wen u miss someone	the person is definitely special for u..... B...	why to miss them	just Keep-in-touch\" gdeve.."
444	ham	\HEY HEY WERETHE MONKEESPEOPLE SAY WE MONKEYAR...	HOWU DOIN? FOUNDURSELF A JOBYET SAUSAGE?LOVE ...	NaN	NaN
671	spam	SMS. ac sun0819 posts HELLO:\You seem cool	wanted to say hi. HI!!!\" Stop? Send STOP to ...	NaN	NaN
710	ham	Height of Confidence: All the Aeronautics prof...	this wont even start........ Datz confidence.."	NaN	NaN

In [6]:
```
df[df['Unnamed: 3'].isnull() == False].head()
```

Out[6]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
95	spam	Your free ringtone is waiting to be collected....	PO Box 5249	MK17 92H. 450Ppw 16"	NaN
281	ham	\Wen u miss someone	the person is definitely special for u..... B...	why to miss them	just Keep-in-touch\" gdeve.."
899	spam	Your free ringtone is waiting to be collected....	PO Box 5249	MK17 92H. 450Ppw 16"	NaN
1038	ham	Edison has rightly said, \A fool can ask more ...	GN	GE	GNT:-)"
2170	ham	\CAN I PLEASE COME UP NOW IMIN TOWN.DONTMATTER...	JUST REALLYNEED 2DOCD.PLEASE DONTPLEASE DONTIG...	U NO THECD ISV.IMPORTANT TOME 4 2MORO\""	NaN

In [7]:

```
df[df['Unnamed: 4'].isnull() == False].head()
```

Out[7]:

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
281	ham	\Wen u miss someone	the person is definitely special for u..... B...	why to miss them	just Keep-in-touch\" gdeve.."
1038	ham	Edison has rightly said, \A fool can ask more ...	GN	GE	GNT:-)"
2255	ham	I just lov this line: \Hurt me with the truth	I don't mind	i wil tolerat.bcs ur my someone..... But	Never comfort me with a lie\" gud ni8 and swe...
3525	ham	\HEY BABE! FAR 2 SPUN-OUT 2 SPK AT DA MO... DE...	HAD A COOL NYTHO	TX 4 FONIN HON	CALL 2MWEN IM BK FRMCLOUD 9! J X\""
4668	ham	When I was born, GOD said, \Oh No! Another IDI...	GOD said	\"OH No! COMPETITION\". Who knew	one day these two will become FREINDS FOREVER!"

In [8]:

```
#the unknown columns are sparsely populated and most that are are populated appear to
#(such as time or address info).  droping these columns
to_drop = ['Unnamed: 2',"Unnamed: 3","Unnamed: 4"]
df = df.drop(columns = to_drop)
print(df.shape)
```

```
(5572, 2)
```

In [9]:

```
#renaming columns
rename_list = {'v1':'label','v2':'documents'}
df = df.rename(columns=rename_list)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   label      5572 non-null   object
 1   documents  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

In [10]:
```
#neither column has any null values, but lets check to make sure there is non-blank te
df_temp = df['documents'].str.len() - df['documents'].str.count(' ')
sum(df_temp == 0)
```

Out[10]:
```
0
```

In [11]:
```
#okay so all the documents contain at least some characters.  Lets check that our labe
label_list = df.label.unique()
print(label_list)
```

```
['ham' 'spam']
```

In [12]:
```
#creating one hotkey on label
label_binary = pd.get_dummies(df.label)
label_binary= label_binary.drop(columns='ham')
label_binary = label_binary.rename(columns={'spam':'label_binary'})
df = pd.concat([df,label_binary],axis=1)
```

In [13]:
```
#checking hotkey join and binary hotkey labeling
print(df.shape)
print(df[df['label']=='ham'].label_binary.unique())
print(df[df['label']=='spam'].label_binary.unique())
```

```
(5572, 3)
[0]
[1]
```

In [14]:
```
#checking for duplicates
df.duplicated().sum()
```

Out[14]:
```
403
```

In [15]:
```
#dropping duplicated
df = df.drop_duplicates()
df.shape
```

Out[15]:
```
(5169, 3)
```

EDA

In [16]:
```
#looking at the frequency of ham versus spam
label_count = df.groupby('label').count()
print(label_count)
```

```
       documents  label_binary
label                         
ham         4516          4516
spam         653           653
```

In [17]:
```
#lets look at how wordy our documents are - first creating a word count
documents = df['documents'].tolist()
word_count = []
for i in documents:
    word_count.append(len(i.split()))
print(len(word_count))
```

```
5169
```

In [18]:
```
#calculating mean, standard deviations, min, and max
min_val = min(word_count)
max_val =max(word_count)
mean_val = np.mean(word_count)
var_val = np.std(word_count)
stat_label = pd.Series(('min','max','mean','std'))
stats = pd.Series((min_val,max_val,mean_val,var_val))
d = {'label':stat_label,'value':stats}
df_stat = pd.DataFrame(data=d)
df_stat
```

Out[18]:

	label	value
0	min	1.000000
1	max	171.000000
2	mean	15.340685
3	std	11.067417

In [19]:
```
#adding the word count into the data frame
df['word_count'] = np.array(word_count)
df.shape
```

Out[19]:
```
(5169, 4)
```

In [20]:
```
#looking at a few of these one word documents
df[df['word_count'] == 1].head()
```

Out[20]:

	label	documents	label_binary	word_count
260	ham	Yup	0	1
275	ham	Thanx...	0	1
283	ham	Okie...	0	1
286	ham	Ok..	0	1
782	ham	Beerage?	0	1

In [21]:
```
#what percentage of the documents have only 1 word
sum(df['word_count'] == 1)/len(df)
```

Out[21]:
```
0.003869220352099052
```

In [22]:
```
range(len(documents))
```

```
Out[22]: range(0, 5169)
```

```
In [23]: #look at the most common words - first prep a word list
         word_list = []
         for i in range(len(documents)):
             word_list.append(documents[i].lower().split())
         master_word_list = list(itertools.chain(*word_list))
```

```
In [24]: #now count the words
         count_words = collections.Counter(master_word_list)
         count_words.most_common(20)
```

```
Out[24]: [('i', 2095),
          ('to', 2055),
          ('you', 1832),
          ('a', 1281),
          ('the', 1223),
          ('and', 919),
          ('u', 890),
          ('in', 785),
          ('is', 766),
          ('my', 676),
          ('for', 653),
          ('your', 618),
          ('me', 579),
          ('of', 552),
          ('have', 532),
          ('on', 476),
          ('call', 468),
          ('are', 457),
          ('that', 453),
          ('it', 440)]
```

Text Preprocessing

```
In [25]: #making text lowercase
         df['documents_clean'] = df['documents'].str.lower()
```

```
In [26]: #replacing URLs with keyword "URL"
         df['documents_clean'] = df['documents_clean'].str.replace(r'https?://\S+|www\.\S+', 'u
```

```
C:\Users\CGLam\AppData\Local\Temp\ipykernel_184\3020070343.py:2: FutureWarning: The d
efault value of regex will change from True to False in a future version.
  df['documents_clean'] = df['documents_clean'].str.replace(r'https?://\S+|www\.\S+',
'url')
```

```
In [27]: #loading stop words
         nltk.download('stopwords')
         stop_words = set(stopwords.words('english'))
         print(len(stop_words))
```

```
179
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\CGLam\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
In [28]: #removing stop words
         df['documents_clean'] = df['documents_clean'].apply(lambda x: ' '.join([word for word
```

In [29]:
```
#remove punctuation
df['documents_clean'] = df['documents_clean'].str.replace(r'[^\w\s]+', '')
```

```
C:\Users\CGLam\AppData\Local\Temp\ipykernel_184\2423228234.py:2: FutureWarning: The d
efault value of regex will change from True to False in a future version.
  df['documents_clean'] = df['documents_clean'].str.replace(r'[^\w\s]+', '')
```

In [30]:
```
#re-reviewing most common words to see if it makes sense to create any custom stop wor
word_list_2 = []
documents_2 = df['documents_clean'].tolist()
for i in range(len(documents_2)):
    word_list_2.append(documents_2[i].lower().split())
master_word_list_2 = list(itertools.chain(*word_list_2))
count_words_2 = collections.Counter(master_word_list_2)
count_words_2.most_common(20)
```

Out[30]:
```
[('u', 1001),
 ('call', 487),
 ('im', 447),
 ('2', 443),
 ('get', 364),
 ('ur', 316),
 ('go', 269),
 ('4', 257),
 ('ltgt', 254),
 ('ok', 251),
 ('free', 243),
 ('know', 239),
 ('got', 231),
 ('like', 231),
 ('good', 217),
 ('come', 210),
 ('ill', 206),
 ('you', 200),
 ('time', 199),
 ('now', 198)]
```

In [31]:
```
#creating custom stop words
custom_stopwords = {'u','im','ur','ill','you'}
```

In [32]:
```
#remove custom stop words
df['documents_clean'] = df['documents_clean'].apply(lambda x: ' '.join([word for word
```

In [33]:
```
#remove non-character tokens
df['documents_clean'] = df['documents_clean'].apply(lambda x: ' '.join([word for word
```

In [34]:
```
#applying stemming
stemmer = PorterStemmer()
df['documents_clean'] = df['documents_clean'].apply(lambda x: ' '.join([stemmer.stem(y
```

In [35]:
```
df.head()
```

Out[35]:

	label	documents	label_binary	word_count	documents_clean
0	ham	Go until jurong point, crazy.. Available only ...	0	20	go jurong point crazi avail bugi n great world...
1	ham	Ok lar... Joking wif u oni...	0	6	ok lar joke wif oni
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	1	28	free entri wkli comp win fa cup final tkt may ...
3	ham	U dun say so early hor... U c already then say...	0	11	dun say earli hor c alreadi say
4	ham	Nah I don't think he goes to usf, he lives aro...	0	13	nah think goe usf live around though

In [36]:
```
#export preprocessed data to excel for further review
#df.to_excel('preprocessed.xlsx')
```

Tokenize the Data

In [37]:
```
def define_tokenizer(x):
    tokenizer = tf.keras.preprocessing.text.Tokenizer()
    tokenizer.fit_on_texts(x)
    return tokenizer

def encode(x2, tokenizer):
    encoded_sentences = tokenizer.texts_to_sequences(x2)
    encoded_sentences = tf.keras.preprocessing.sequence.pad_sequences(encoded_sentence
    return encoded_sentences
```

In [38]:
```
tokenizer = define_tokenizer(df['documents_clean'])
s_strings = encode(df['documents_clean'],tokenizer)
```

In [39]:
```
#checking that we have appropriate number of documents
len(s_strings)
```

Out[39]:
```
5169
```

In [40]:
```
#quick look at encoding...text of first clean document
df['documents_clean'][0]
```

Out[40]:
```
'go jurong point crazi avail bugi n great world la e buffet cine got amor wat'
```

In [41]:
```
#encoding of that document
s_strings[0]
```

Out[41]:
```
array([   2, 2952,  271,  540,  568,  954,   43,   66,  325,  955,   88,
       2089,  956,   11, 2953,   64,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0])
```

In [42]:
```
#pulling these words out of the dictionary to make sure we encoded as expected
d = tokenizer.word_index
print(d['go'])
print(d['jurong'])
print(d['point'])
print(d['crazi'])
print(d['avail'])
print(d['bugi'])
print(d['n'])
print(d['great'])
print(d['world'])
print(d['la'])
print(d['e'])
print(d['buffet'])
print(d['cine'])
print(d['got'])
print(d['amor'])
print(d['wat'])
```

```
2
2952
271
540
568
954
43
66
325
955
88
2089
956
11
2953
64
```

One Hotkey Encoding (Count Vectorization)

In [43]:
```
#creating the one hotkey on clean documents
vec = CountVectorizer()
X_train_count = vec.fit_transform(df['documents_clean'].values)
X_train_count.toarray()
```

Out[43]:
```
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=int64)
```

In [44]:
```
#moving into pandas dataframe
df_one = pd.DataFrame(X_train_count.toarray())
len(df_one)
```

Out[44]:
```
5169
```

In [45]:
```
df_one['y'] = df['label_binary'].tolist()
```

Train Test Split

In [46]:
```
#seperate depedendent and indepedent variables
y = df_one['y']
x = df_one.drop(columns=['y'])
```

In [47]:
```
#creating train/test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.5, random_state=
```

In [48]:
```
#checking shape of test train splits
print("Train Independent Variable Shape:",x_train.shape)
print("Test Independent Variable Shape:",x_test.shape)
print("Train Dependent Variable Shape:",y_train.shape)
print("Test Dependent Variable Shape:",y_test.shape)
```

```
Train Independent Variable Shape: (2584, 6793)
Test Independent Variable Shape: (2585, 6793)
Train Dependent Variable Shape: (2584,)
Test Dependent Variable Shape: (2585,)
```

In [49]:
```
#export
x_train.to_csv("Data/x_train.csv",index=False)
x_test.to_csv("Data/x_test.csv",index=False)
y_train.to_csv("Data/y_train.csv",index=False)
y_test.to_csv("Data/y_test.csv",index=False)
```