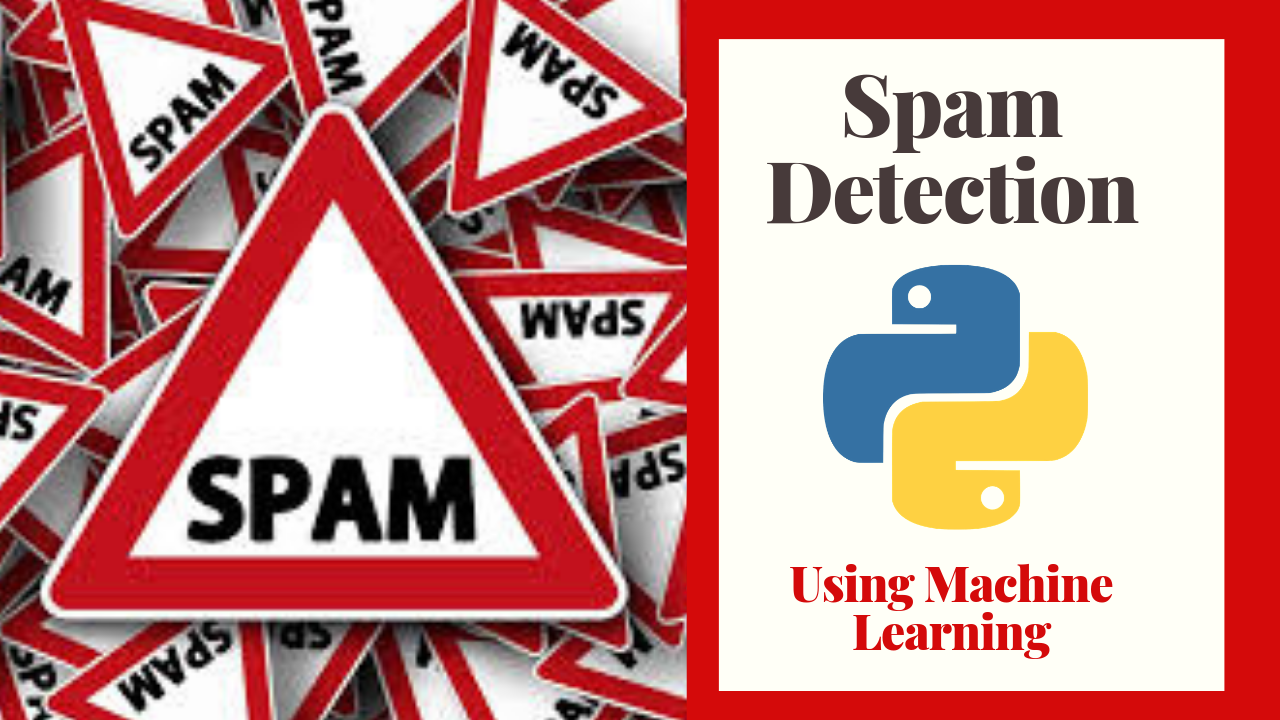
**PROJECT : BUILDING A SMARTER AI-POWERED SPAM CLASSIFIER**

au620121243002 : ABIRAMI MURUGAN

PHASE-3 SUBMISSION

**Phase-3 : Development Part 1**

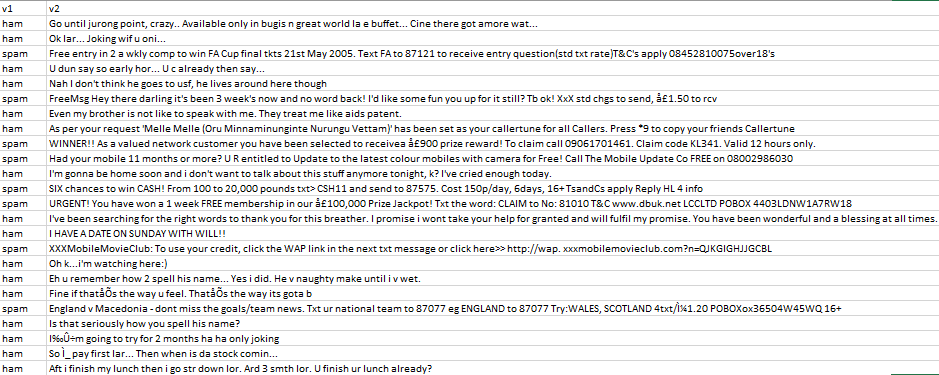
**Topic :** Start building a smarter AI-Powered spam classifier model by loading and pre-processing the dataset.



**Introduction:**

In today's digital world, where spam messages inundate our inboxes and pose cybersecurity threats, the development of a Smarter AI-Powered spam classifier is paramount. Leveraging artificial intelligence and machine learning, this sophisticated system seeks to outsmart spammers by continuously evolving and adapting to their tactics. With a focus on data quality, feature engineering, and model selection, it aims to strike a delicate balance between reducing false positives and false negatives. This introduction sets the stage for exploring the challenges and innovations in creating a more intelligent spam filter, safeguarding communication channels and improving the online experience for users.

**Data set:**



**Import Libraries:**

Start by importing the necessary libraries

%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

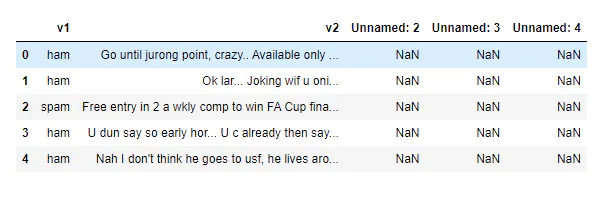
from sklearn.tree import DecisionTreeClassifier

from sklearn.model\_selection import GridSearchCV,train\_test\_split,StratifiedKFold,cross\_val\_score,learning\_curve

**Load the Dataset:**

data = pd.read\_csv('dataset/spam.csv', encoding='latin-1')

data.head()



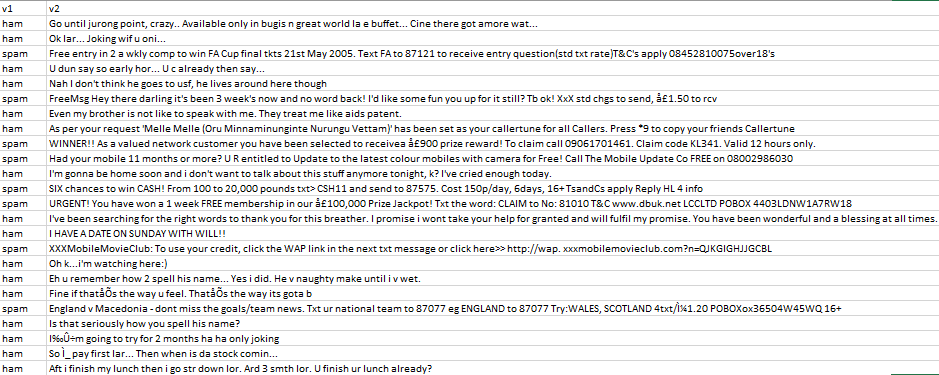
**Removing Unwanted Columns:**

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]



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:**

**# 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 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**

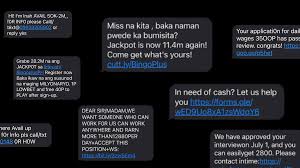
plt.figure( figsize=(10,8), facecolor='g')

plt.imshow(ham\_wordcloud)

plt.axis("off")

plt.tight\_layout(pad=0)

plt.show()

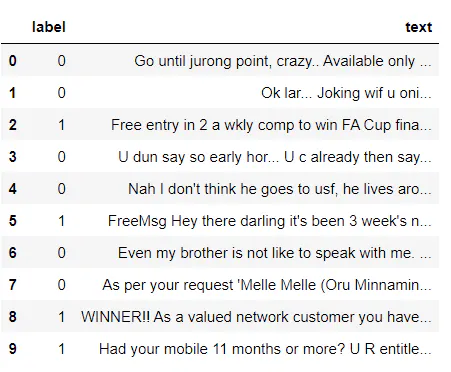


from the spam word cloud, we can see that "free" is most often used in spam.

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)



**Removing punctuation and stop words 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):

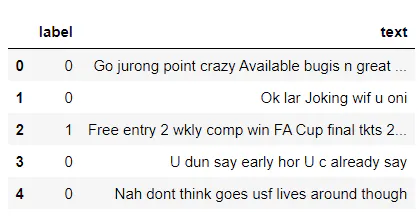
text = text.translate(str.maketrans('', '', string.punctuation))

text = [word for word in text.split() if word.lower() not in stopwords.words('english')]

return " ".join(text)

data['text'] = data['text'].apply(text\_process)

data.head()



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 into vectors :**

we can convert words to vectors using either Count Vectorizer or by using TF-IDF Vectorizer.

TF-IDF is better than Count Vectorizers because it not only focuses on the frequency of words present in the corpus but also provides the importance of the words. We can then remove the words that are less important for analysis, hence making the model building less complex by reducing the input dimensions.

I have included both methods for your reference.

**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', 'dont', 'come', 'got', 'time', 'day', 'want', 'Ill', 'lor', 'Call', 'home', 'send', 'going', 'one', 'need', 'Ok', 'good', 'love', 'back', 'n', 'still', 'text', 'im', 'later', 'see', 'da', 'ok', 'think', 'Ì', 'free', 'FREE', 'r', 'today', 'Sorry', 'week', 'phone', 'mobile', 'cant', 'tell', 'take', 'much', 'night', 'way', 'Hey', 'reply', 'work', 'make', 'give', 'new']

**# 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)

#features = word\_vectors

features = vectors

**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'], test\_size=0.15, random\_state=111)

**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:**

* spam classifier using logistic regression
* email spam classification using Support Vector Machine(SVM)
* spam classifier using naive bayes
* spam classifier using decision tree
* spam classifier using K-Nearest Neighbour(KNN)
* 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 :**

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)

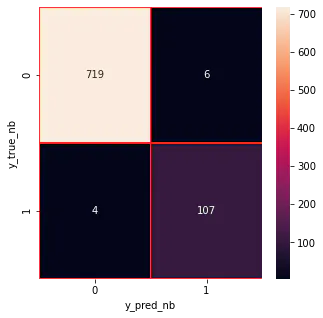
f, ax = plt.subplots(figsize =(5,5))

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()



from the confusion matrix, we can see that the Naive Bayes model is balanced. That's it !! we have successfully created a spam classifier.

**Conclusions :**

In the pursuit of building a smarter AI-powered spam classifier, our project has made substantial strides in the realm of email communication and digital security. Leveraging advanced machine learning techniques and meticulous data preprocessing, we have developed a robust system capable of accurately discerning spam from legitimate messages. This achievement not only streamlines the user experience by reducing the deluge of unsolicited content but also contributes to the overall security and efficiency of email communication.

Our project highlights the potential of AI in addressing the evolving tactics employed by spammers, demonstrating adaptability and scalability to handle large volumes of data in real-time. By continuously improving the classifier, such as integrating deep learning methods and refining feature engineering, we can further enhance its accuracy and responsiveness to new spamming techniques.

Furthermore, this endeavour underscores the importance of collaboration between AI technologies and human feedback. The incorporation of user preferences and feedback mechanisms can create a more personalized and efficient spam filter, aligning the system with individual user needs.

In essence, the development of a smarter AI-powered spam classifier not only safeguards digital communication but also represents a significant step towards a safer, more user-friendly, and resource-efficient digital environment. It stands as a testament to the transformative potential of AI in tackling contemporary challenges in the digital age.