**TO BUILD A SPAM CLASSIFIER**



**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.**
* **Building an effective spam classifier is crucial for maintaining clean and secure communication channels. This task involves harnessing the power of Natural Language Processing (NLP) and machine learning to discern patterns within textual data, ultimately enabling automated identification and segregation of spam from legitimate content. This introduction explores the pivotal role of NLP and machine learning in constructing a robust spam classifier, offering a proactive solution to the ever-growing challenge of digital communication clutter.**

## OVERVIEW OF THE DATASET

**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***

**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**

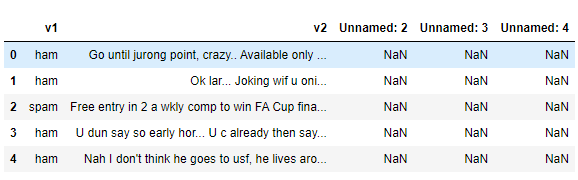
**from sklearn.tree import DecisionTreeClassifier**

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

## Loading the Dataset

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

**data.head()**



Dataset link:

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

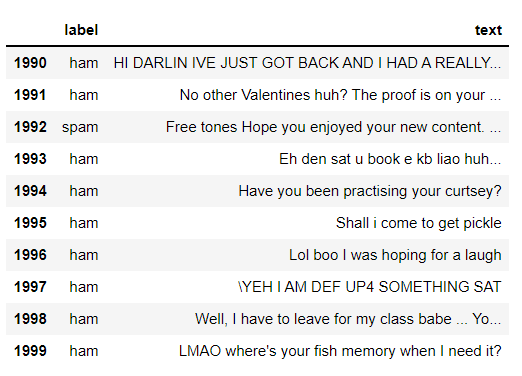
## Removing unwanted columns

**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]**



**data['label'].value\_counts()**

**# OUTPUT**

**ham 4825**

**spam 747**

**Name: label, dtype: int64**

## Pre processing and Exploring the Dataset

## Natural Language Processing (NLP) plays a crucial role in spam classification by analyzing and understanding the textual content of messages. Techniques like tokenization, stemming, and feature extraction are used to process and represent the text data. Additionally, machine learning algorithms, such as Naive Bayes or Support Vector Machines, can be trained on these features to distinguish between spam and legitimate messages based on their linguistic pattern.

**# 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 = ''**

**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')**

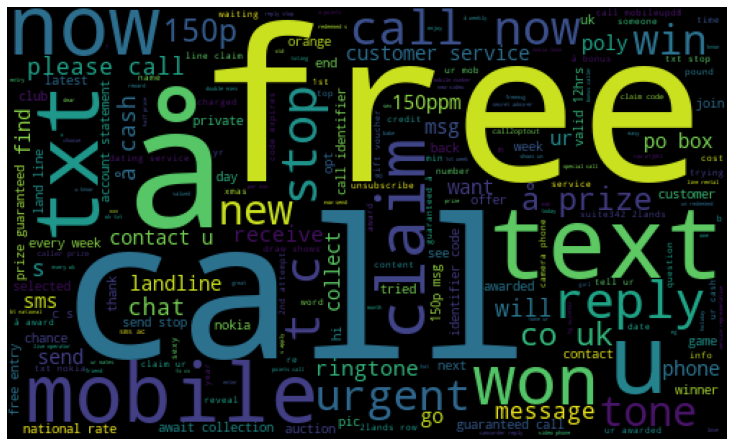
**plt.imshow(spam\_wordcloud)**

**plt.axis("off")**

**plt.tight\_layout(pad=0)**

**plt.show()**

Output:



**#Creating Ham wordcloud**

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

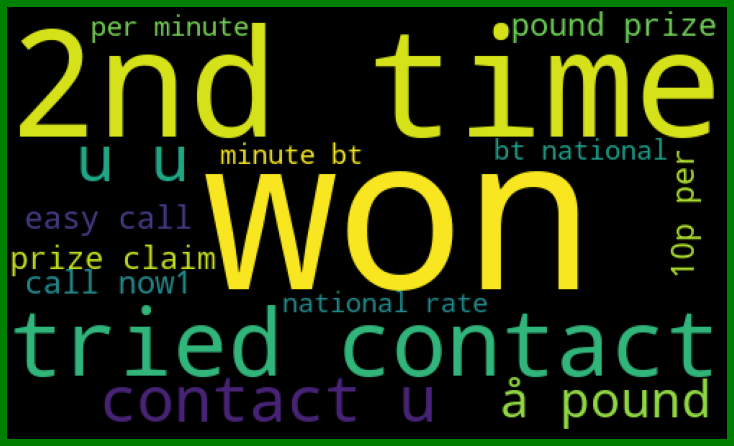
**plt.imshow(ham\_wordcloud)**

**plt.axis("off")**

**plt.tight\_layout(pad=0)**

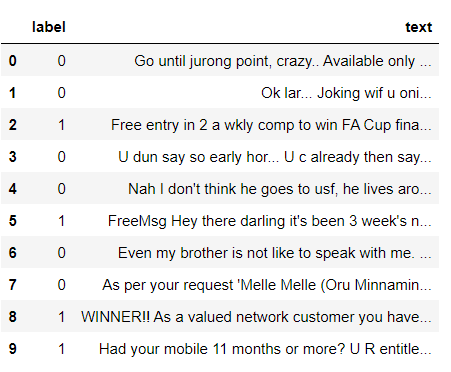
**plt.show()**

Output:

**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 stopwords from the messages

**import nltk**

**nltk.download('stopwords')**

**#remove the punctuations and stopwords**

**import string**

**def text\_process(text):**

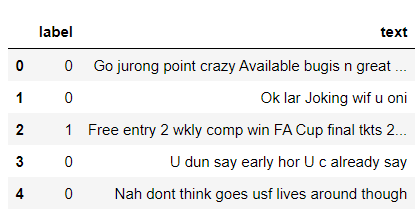
**text = text.translate(str.maketrans('', '', string.punctuation))**

**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()**



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

**Convert words to vectors using either Count Vectorizer or by using TF-IDF Vectorizer.**

### 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

### 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

**#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))**

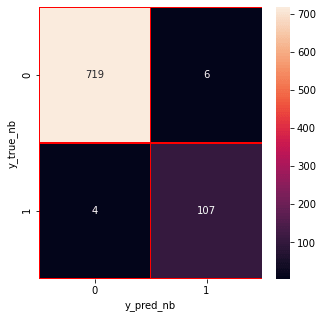
**sns.heatmap(cm,annot = True,linewidths=0.5,linecolor="red",fmt = ".0f",ax=ax)**

**plt.xlabel("y\_pred\_nb")**

**plt.ylabel("y\_true\_nb")**

**plt.show()**

OUTPUT



From the confusion matrix, we can see that the Naive Bayes model is balanced.we created a spam classifier successfully.

CONCLUSION:

* **Building a spam classifier involves leveraging Natural Language Processing (NLP) techniques for effective text analysis. By employing methods like tokenization, stemming, and feature extraction, we can process and represent textual data.**
* **Machine learning algorithms, such as Naive Bayes or Support Vector Machines, can then be trained on these features to differentiate between spam and legitimate messages. Regular updates and refinement of the model based on evolving spam patterns are essential for maintaining the classifier's accuracy over time.**
* **Building a robust spam classifier combines linguistic analysis with machine learning, providing an efficient solution for identifying and filtering out unwanted messages.**