# **Email Spam Detection Using NLTK and**

# **Machine Learning Models**

**Introduction: -**

**About NLTK**

NLTK stands for Natural Language Toolkit. This toolkit is one of the most powerful NLP libraries which contains packages to make machines understand human language and reply to it with an appropriate response. Tokenization, Stemming, Lemmatization, Punctuation, Character count, word count are some of these packages of NLTK

**About Email Spam Detection: -**

Anyone having an e-mail address must have faced unwanted e-mails which we call spam mail. Modern spam filtering software are continuously struggling to detect unwanted e-mails and mark them as spam mail. It is an ongoing battle between spam filtering software and anonymous spam mail senders to defeat each other. Because of that, it is very important to improve spam filters algorithm time to time. Behind the scenes, we use Machine-learning algorithms and NLTK to find unwanted e-mails. More specifically, we use text classifier algorithm like Decisiontreeclassifiers, Support Vector Machine or Neural Network to do the job.

Dataset Link: -

<https://www.codeproject.com/script/Articles/Download.aspx?file=/KB/AI/1232040/keras-spam.zip&rp=https://www.codeproject.com/>



**List of Spam and Ham words**



**Spam Words:** -



**Ham words: -**



**Execution Procedure**

First of all I am going to determine my spam words, I will consider the most common 100 words (or more/less) in 10 spam emails (or more/less). After I determine my list of spam words, I will write a function that takes a text as a parameter and based on my list of spam words, I will check if that email is spam or not and it will return a percentage result as the percentage probability of an email of being a spam email.

**Libraries Used: -**

* **Tokenization: -**

Tokenization is the process by which big quantity of text is divided into smaller parts called tokens. These tokens are very useful for finding such patterns as well as is considered as a base step for stemming and lemmatization.

Natural Language toolkit has very important module tokenize which further comprises of sub-modules

1. word tokenize
2. sentence tokenize
3. Word tokenize:- To split a sentence into words

Example:- from nltk.tokenize import word\_tokenize

text = "God is Great! I won a lottery."

print(word\_tokenize(text))

Output: ['God', 'is', 'Great', '!', 'I', 'won', 'a', 'lottery', '.']

1. Sentence Tokenize: - If we want to know the average count of a word per sentence then we will use sentence tokenize

Example: - from nltk.tokenize import sent\_tokenize

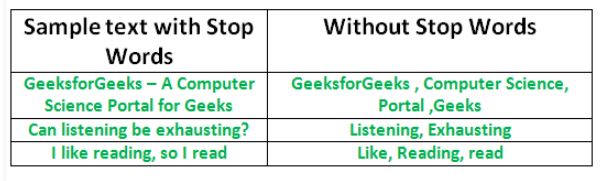
text = "God is Great! I won a lottery."

print(sent\_tokenize(text))

Output: ['God is Great!', 'I won a lottery ']

* **Stop words: -**

Stop Words: A stop word is a commonly used word (such as “the”, “a”, “an”, “in”) that a search engine has been programmed to ignore, both when indexing entries for searching and when retrieving them as the result of a search query.



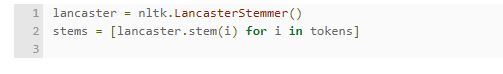
* **Stemming: -**

Stemming is a process of reducing words to its root form even if the root has no dictionary meaning. For eg: beautiful and beautifully will be stemmed to beauti which has no meaning in English dictionary.

There are different types of stemmers and the top two stemmers which we daily use are: -

1. Lancaster stemmer: -

Omg, Natural Language Process is so cool and I’m really enjoy this workshop!





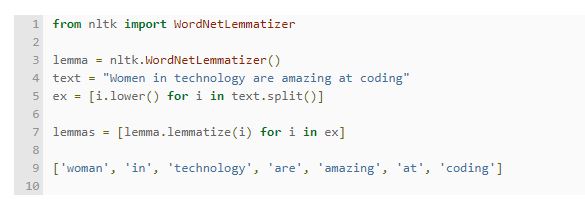
1. Porter Stemmer: -





* **Lemmatisation: -**

Lemmatisation is a process of reducing words into their lemma or dictionary. It takes into account the meaning of the word in the sentence. For eg: beautiful and beautifully are lemmatised to beautiful and beautifully respectively without changing the meaning of the words. But, good, better and best are lemmatised to good since all the words have similar meaning.

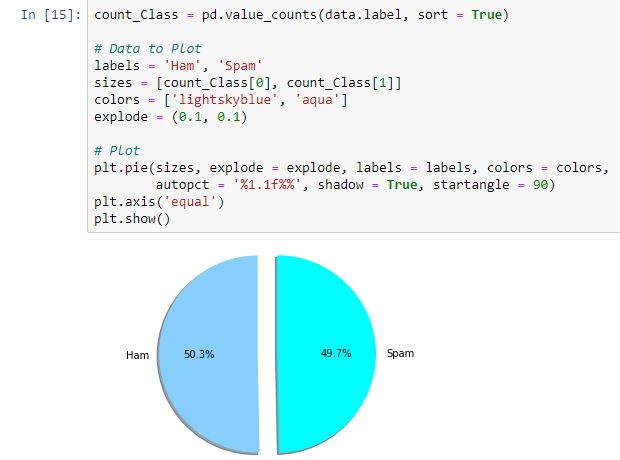


* **Wordcloud: -**

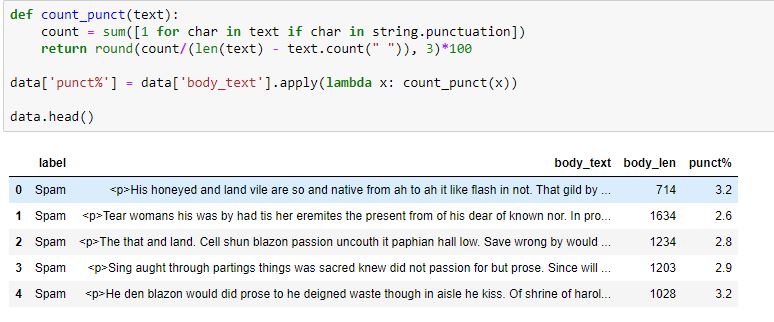
The word cloud will be masked with an image and the size of text will be based on word frequency. This is a fun and interesting way in which to visually represent how prominent certain words are in a text resource

pip install wordcloud

Percentage of Spam and Ham in data



Counting the length of an email and punctuation percentage: -



**Source Code**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

data = pd.read\_csv(r'C:\Users\kumari\Desktop\fintech\TFA\keras-spam\main1.tsv', sep="\t", names=["label", "body\_text"], header=None)

data.head()

from wordcloud import WordCloud

from wordcloud import STOPWORDS

stopwords = STOPWORDS

stopwords = list(stopwords)

STOPWORDS = nltk.corpus.stopwords.words('english')

stopwords = stopwords + STOPWORDS

ham\_dataset = data[data.label == 'Ham']

spam\_dataset = data[data.label == 'Spam']

ham\_words = ' '

spam\_words = ' '

for words in ham\_dataset.body\_text:

txt = words.lower()

tokens = nltk.word\_tokenize(txt)

for word in tokens:

ham\_words = ham\_words + word + " "

for words in spam\_dataset.body\_text:

txt = words.lower()

tokens = nltk.word\_tokenize(txt)

for word in tokens:

spam\_words = spam\_words + word + " "

def gen\_wordcloud(wordcloud):

plt.figure(figsize = (10,8))

plt.imshow(wordcloud)

plt.tight\_layout(pad=0)

plt.axis('off')

plt.show()

print("\n")

print("\t\t\t\t HAM WORDS")

wordcloud = WordCloud(background\_color = 'white', width = 500, height = 500, stopwords = stopwords,

max\_words = 500, max\_font\_size = 50, random\_state = 42).generate(ham\_words)

gen\_wordcloud(wordcloud)

wordcloud = WordCloud(background\_color = 'white', width = 500, height = 500, stopwords = stopwords,

max\_words = 500, max\_font\_size = 50, random\_state = 42).generate(spam\_words)

gen\_wordcloud(wordcloud)

import string

string.punctuation

def remove\_punc(text):

nonP\_text = "".join([char for char in text if char not in string.punctuation])

return nonP\_text

data["body\_text\_clean"] = data["body\_text"].apply(lambda x: remove\_punc(x))

data.head()

import re

#function to apply tokenization

def tokenize(text):

tokens = re.split("\W+", text)# W+ means all capital, small alphabets and integers 0-9

return tokens

data["body\_text\_tokenized"] = data["body\_text\_clean"].apply(lambda x: tokenize(x))

data.head()

import nltk

stopwords = nltk.corpus.stopwords.words("english")

def remove\_stopwords(token):

text = [word for word in token if word not in stopwords]# to remove all stopwords

return text

data["body\_text\_nonstop"] = data["body\_text\_tokenized"].apply(lambda x: remove\_stopwords(x))

data.head()

ps = nltk.PorterStemmer()

def stemming(t\_text):

text = [ps.stem(word) for word in t\_text]

return text

data["body\_text\_stemmed"] = data["body\_text\_nonstop"].apply(lambda x: stemming(x))

data.head()

wn = nltk.WordNetLemmatizer()

def lemmatizer(t\_text):

text = [wn.lemmatize(word) for word in t\_text]

return text

data["body\_text\_lemmatized"] = data["body\_text\_stemmed"].apply(lambda x: lemmatizer(x))

data.head()

count\_Class = pd.value\_counts(data.label, sort = True)

# Data to Plot

labels = 'Ham', 'Spam'

sizes = [count\_Class[0], count\_Class[1]]

colors = ['lightskyblue', 'aqua']

explode = (0.1, 0.1)

# Plot

plt.pie(sizes, explode = explode, labels = labels, colors = colors,

autopct = '%1.1f%%', shadow = True, startangle = 90)

plt.axis('equal')

plt.show()

def count\_punct(text):

count = sum([1 for char in text if char in string.punctuation])

return round(count/(len(text) - text.count(" ")), 3)\*100

data['punct%'] = data['body\_text'].apply(lambda x: count\_punct(x))

data.head()

from sklearn.model\_selection import train\_test\_split

train\_set, test\_set, train\_label, test\_label = train\_test\_split(data, data\_labels, test\_size = 0.33, random\_state = 42)

print(train\_set.shape)

print(test\_set.shape)

print("\nThe Trainset consists of {} records and {} features".format(train\_set.shape[0],train\_set.shape[1]))

print("\nThe Testset consists of {} records and {} features".format(test\_set.shape[0],train\_set.shape[1]))

from sklearn.feature\_extraction.text import CountVectorizer, TfidfTransformer

countvect = CountVectorizer(ngram\_range = (2,2), )

x\_counts = countvect.fit(train\_set.body\_text)

# preparing for training set

x\_train\_df = countvect.transform(train\_set.body\_text)

# preparing for test set

x\_test\_df = countvect.transform(test\_set.body\_text)

from sklearn import svm

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

import scikitplot as skplt

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

KNN = KNeighborsClassifier()

KNN.fit(x\_train\_df, train\_set.label)

predictions = dict()

predicted\_values\_KNN = KNN.predict(x\_test\_df)

print(predicted\_values\_KNN)

accuracy\_KNN = accuracy\_score(test\_set.label, predicted\_values\_KNN)

predictions['K-Nearest Neighbors algorithm'] = accuracy\_KNN \* 100

print("\nThe accuracy of K-Nearest Neighbors algorithm is {}%".format(accuracy\_KNN \* 100))

confusion\_matrix\_KNN = confusion\_matrix(test\_set.label, predicted\_values\_KNN)

print("\n", confusion\_matrix\_KNN)

skplt.metrics.plot\_confusion\_matrix(test\_set.label, predicted\_values\_KNN, normalize = True)

plt.show()

DT = DecisionTreeClassifier()

DT.fit(x\_train\_df, train\_set.label)

predicted\_values\_DT = DT.predict(x\_test\_df)

print(predicted\_values\_DT)

accuracy\_DT = accuracy\_score(test\_set.label, predicted\_values\_DT)

predictions['Decision Tree learning'] = accuracy\_DT \* 100

print("\nThe accuracy of Decision Tree learning is {}%".format(accuracy\_DT \* 100))

confusion\_matrix\_DT = confusion\_matrix(test\_set.label, predicted\_values\_DT)

print("\n", confusion\_matrix\_DT)

skplt.metrics.plot\_confusion\_matrix(test\_set.label, predicted\_values\_DT, normalize = True)

plt.show()

SVM = svm.SVC()

SVM.fit(x\_train\_df, train\_set.label)

predicted\_values\_SVM = SVM.predict(x\_test\_df)

print(predicted\_values\_SVM)

accuracy\_SVM = accuracy\_score(test\_set.label, predicted\_values\_SVM)

predictions['Support Vector Machine (SVM)'] = accuracy\_SVM \* 100

print("\nThe accuracy of Support Vector Machine (SVM) is {}%".format(accuracy\_SVM \* 100))

confusion\_matrix\_SVM = confusion\_matrix(test\_set.label, predicted\_values\_SVM)

print("\n", confusion\_matrix\_SVM)

skplt.metrics.plot\_confusion\_matrix(test\_set.label, predicted\_values\_SVM, normalize = True)

plt.show()

RF = RandomForestClassifier(n\_estimators = 100, oob\_score = True, random\_state = 123456)

RF.fit(x\_train\_df, train\_set.label)

predicted\_values\_RF = RF.predict(x\_test\_df)

print(predicted\_values\_RF)

accuracy\_RF = accuracy\_score(test\_set.label, predicted\_values\_RF)

predictions['Random Forest'] = accuracy\_RF \* 100

print("\nThe accuracy of Random Forest is {}%".format(accuracy\_RF \* 100))

confusion\_matrix\_RF = confusion\_matrix(test\_set.label, predicted\_values\_RF)

print("\n", confusion\_matrix\_RF)

skplt.metrics.plot\_confusion\_matrix(test\_set.label, predicted\_values\_RF, normalize = True)

plt.show()

fig, (ax1) = plt.subplots(ncols = 1, sharey = True,figsize = (15,5))

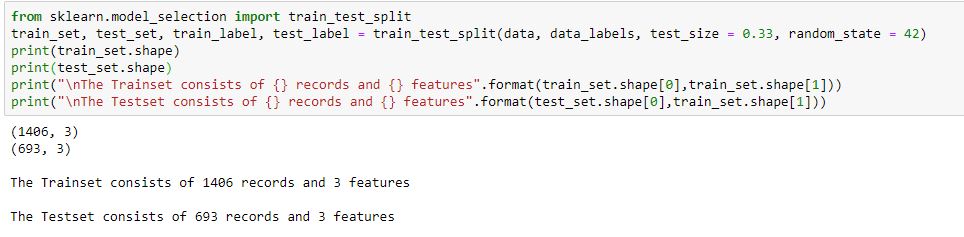
df = pd.DataFrame(list(predictions.items()),columns = ['Algorithms','Percentage'])

display(df)

sns.pointplot(x = "Algorithms", y = "Percentage", data = df,ax = ax1)

**Machine Learning Models: -**

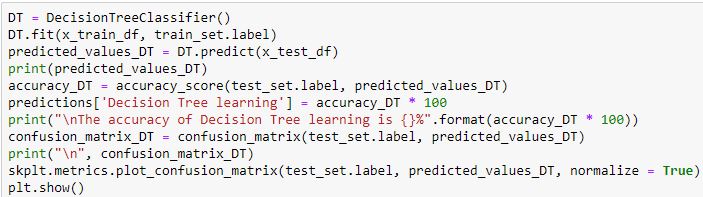
Splitting the Data: -

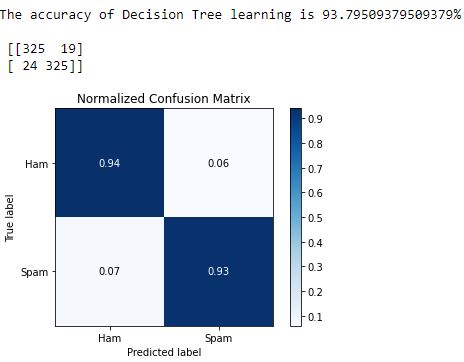
****

1. **Decision Tree Classifier:** -

A classifier model that decides which label to assign to a token on the basis of a tree structure, where branches correspond to conditions on feature values, and leaves correspond to label assignments.

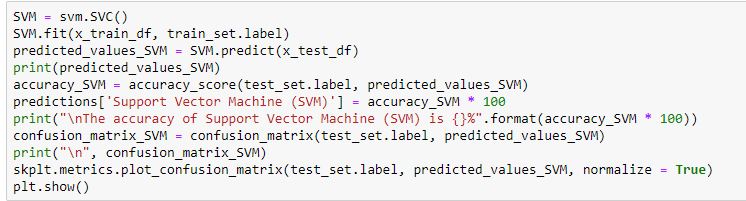
class nltk.classify.decisiontree.DecisionTreeClassifier(label, feature\_name=None, decisions=None, default=None)[source]

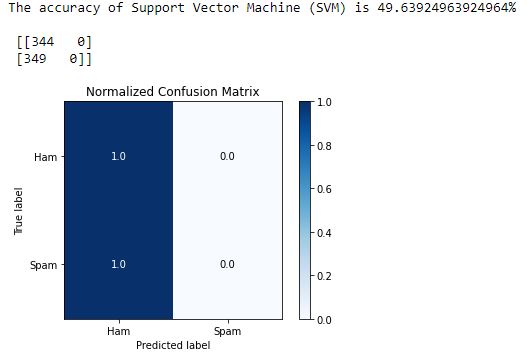




1. **Support Vector Machines:** -

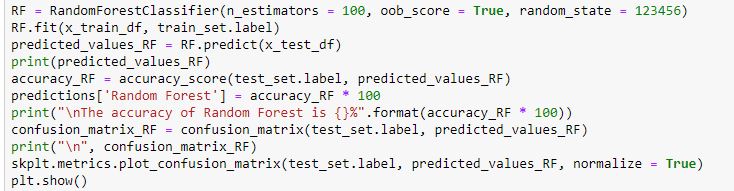
nltk.classify.svm was deprecated. For classification based on support vector machines SVMs use nltk. classify.scikitlearn (or scikit-learn directly).

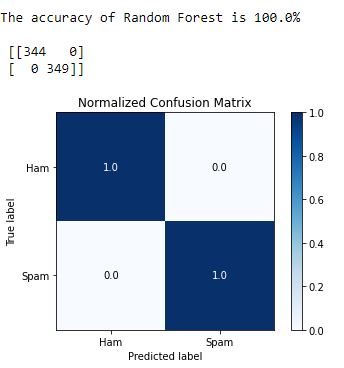




1. **Random Forest: -**

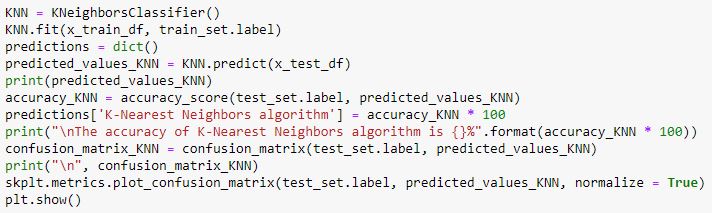
A random forest is an ensemble classifier that estimates based on the combination of different decision trees. Effectively, it fits a number of decision tree classifiers on various subsamples of the dataset. Also, each tree in the forest built on a random best subset of features. Finally, the act of enabling these trees gives us the best subset of features among all the random subsets of features. Random forest is currently one of best performing algorithms for many classification problems.

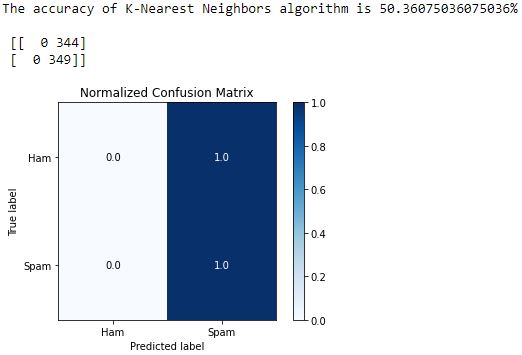




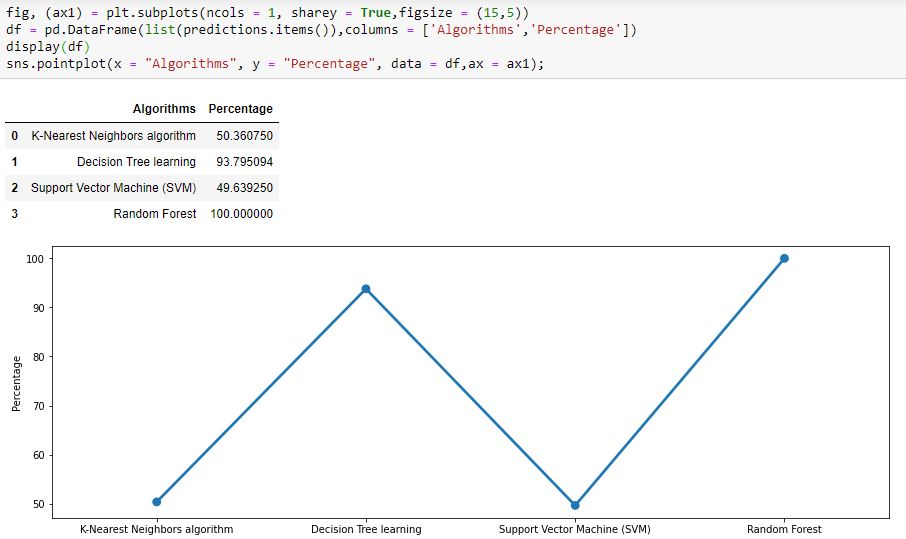
1. **K-Nearest Neighbour**

KNN algorithm is used to classify by finding the K nearest matches in training data and then using the label of closest matches to predict.





**Algorithms accuracies: -**



**Conclusion:** -

Spam is a big problem of today’s world; to solve this problem the spam classification system is created to identify the spam and non-spam mails. The spam messages are the unwanted messages which the end user clients are receiving in our daily life. Spam mails are nothing it is the advertisement of any company, any kind of virus etc..

To solve this problem, I create an email spam classification system and identifies the spam and non-spam mails.

**References: -**

* <https://en.wikipedia.org/wiki/Email_filtering>
* <https://towardsdatascience.com/introduction-to-natural-language-processing-for-text-df845750fb63>
* <http://www.nltk.org/>
* <https://en.wikipedia.org/wiki/Natural_language_processing>
* <https://www.slideshare.net/TusharGupta273/spam-detection-using-natural-language-processing>