

**PROJECT STATUS**

**CIS 593 BIG DATA**

**Name: Varun Murthy Mokarala**

**CSU ID: 2756232**

**Yeswanth Kumar Varada**

**CSU ID: 2746369**

**Problem Summary:**

**In our daily life we get to know a lot of people. As there are lot of means of communication to communicate with them. Here the electronic mails or the messages play a vital role to communicate with the people. As it makes easier to send the messages. As the technology increases, we receive a lot of spam emails or the messages. Blacklisting, detecting bulk emails, scanning message headings are the different approaches used for the detection of the spam emails or the messages. To have the better effectiveness and also to detect the Ham we use the latest technique content-based spam filtering. In machine learning point of view when we choose spam filtering approach, we can see classification problem. we are supposed to implement a model to detect weather a message is spam or ham. We are going to do this by using classifiers.**

**Software Installations:**

**Anaconda Navigator**

**spyder**

**Classifiers Used:**

**Naïve bayes**

**SVM**

**Procedure and Design:**

In machine learning point of view when we choose spam filtering approach, we can see classification problem. Here we classify email as spam or not spam (Ham) which depends on the features. But, in our point of view the features are the count of each word in the email. Training and Testing are the two modes which a machine learning system operates. In training the AI system is a given labelled data from the training data set. But in our project large set of emails are the labelled training data which is labeled spam or not spam (Ham). In the training process the classifiers prepare from the training data shows the interlink between the email and its label. But in the testing process an unlabeled data is given to the machine learning system. These data are the emails without the spam/ham label. The classifier in the testing shows us either it is spam or ham which depends on the features of an email. In our project we have given the data sets which consist of 5570 messages in which 4845 ham and 725 spam. Now we differentiate spam and ham messages. Pre-processing is the principal stage in which the unstructured information is changed over into progressively organized information. Since watchwords in SMS instant messages are inclined to be supplanted by images. In this investigation, the stop word list remover for English language have been applied to dispose of the stop words in the SMS instant messages. Fig. 2(a) shows the frequencies of words in messages, while Fig. 2(b) shows the most regular words utilized in spam instant messages are from pronoun (for example to) and prepositions (for example your) gatherings. Thus, the top words in ham instant message are involved by either pronoun, prepositions among numerous different sorts of stop words.

Fig.2(a)

A picture containing fence

Description automatically generated

Fig.2(b)

A screenshot of a cell phone

Description automatically generated

**Implementation:**

After doing the pre-processing work i.e., collecting all the most frequently used words in spam and ham separately. Next, we will implement this data using two classifiers that are Naïve bayes and SVM

**Naïve Bayes**

The Naive Bayesian classifier takes its foundations in the well-known Bayes Theorem:

Bayes Theorem basically portrays the amount we ought to change the likelihood that our theory (H) will happen, given some new proof (e).

For our task, we need to decide the likelihood that an email is spam, given the proof of the email's highlights (F1,F2,...Fn). These highlights F1, F2,...Fn are only a Boolean worth (0 or 1) portraying whether the stem 1 through n shows up in the email. At that point, we think about P(Spam|F1,F2,...Fn) to P(Ham|F1,F2,...Fn) and figure out which is more probable. Spam and Ham are viewed as the classes, which are spoken to in the conditions underneath as "C".

Note that when we are looking at P(Spam|F1,F2,...Fn) to P(Ham|F1,F2,...Fn), the denominators are the equivalent. Consequently, we can rearrange the examination by noticing that:

Computing P(F1,F2,...Fn|C) can get monotonous. In this way, the Naive Bayesian classifier makes a suspicion that definitely improves the figures. The supposition that will be that the probabilities of F1, F2, ..Fn happening are generally autonomous of one another. While this supposition that isn't accurate (as certain words are bound to show up together), the classifier despite everything performs very much given this suspicion. Presently, to decide if an email is spam or ham, we simply select the class (C = spam or C = ham) that expands the accompanying condition:

**Support Vector Machine (SVM)**

Vector Machine is utilized for grouping and furthermore for relapse issues where the datasets are utilized to prepare the SVM to group any new information that it gets. It is a directed AI calculation that works by finding a hyperplane that groups the dataset into various classes. The SVM amplifies the separation between various classes on account of the presence of numerous straight hyperplanes which is called as edge expansion.

The help vector machines (SVM) is anything but difficult to utilize and incredible for information characterization. When creating channel model, we make a vector for every datum in that preparing corpus and afterward SVM will delineate vectors into hyperplane. At that point SVM will realize which sorts of vector are close to which class. Here SVM is a decent moving toward technique in definitely discovering best characterization hyper-plans to augment the edge with the goal that we could arrange another message into spam or ham.

Here and there, a direct classifier isn't sufficiently perplexing

From "Bonehead's Guide": Map information into a more extravagant element space including nonlinear highlights, at that point build a hyperplane in that space with the goal that every single other condition are the equivalent Preprocess the information utilizing a change At that point, utilize a classifier

Before our decision rule was of the form:

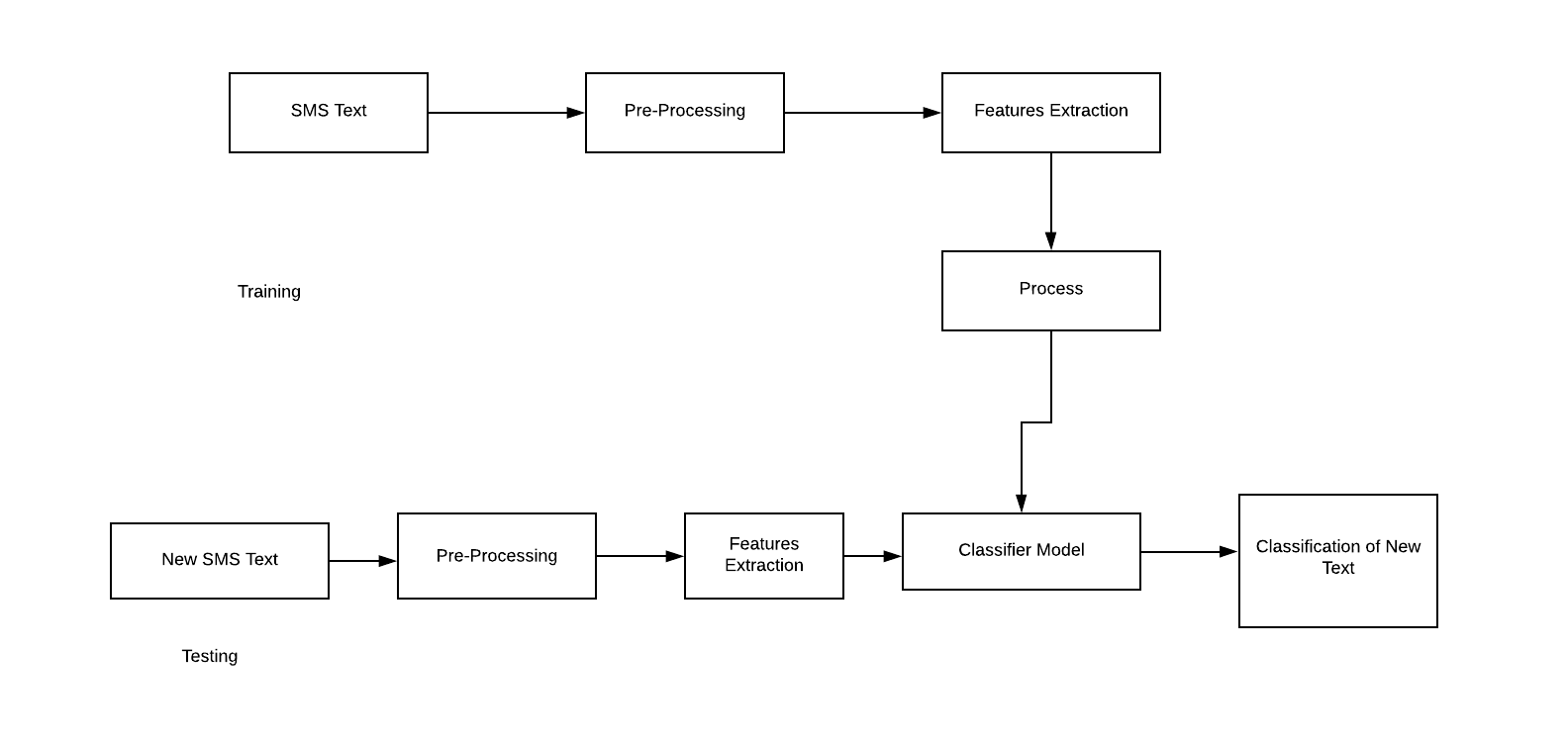
Now, we can write it as:

Kernel Function is

The most applicable machine learning algorithm for our concern is linear svc. Before jumping into linear svc with our information, we're going to show a straight forward model that should help set your comprehension of working with linear svc.

The goal of a linear svc (support vector classifier) is to fit to the information you give, restoring a "best fit" hyperplane that partitions, or orders, your information. From that point, in the wake of getting the hyperplane, you would then be able to take care of certain highlights to your classifier to perceive what the "anticipated" class is. This makes this particular calculation somewhat appropriate for our utilizations, however you can utilize this for some circumstances.

**FLOWCHART:**



**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from collections import Counter

from sklearn import feature\_extraction, model\_selection, naive\_bayes, metrics, svm

from IPython.display import Image

import warnings

warnings.filterwarnings("ignore")

#matplotlib inline

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

data.head(n=10)

count\_Class=pd.value\_counts(data["v1"], sort= True)

count\_Class.plot(kind= 'bar', color= ["green", "red"])

plt.title('Bar chart')

plt.show()

count1 = Counter(" ".join(data[data['v1']=='ham']["v2"]).split()).most\_common(20)

df1 = pd.DataFrame.from\_dict(count1)

df1 = df1.rename(columns={0: "words in Ham", 1 : "count"})

count2 = Counter(" ".join(data[data['v1']=='spam']["v2"]).split()).most\_common(20)

df2 = pd.DataFrame.from\_dict(count2)

df2 = df2.rename(columns={0: "words in spam", 1 : "count\_"})

df1.plot.bar(legend = False, color = 'green')

y\_pos = np.arange(len(df1["words in Ham"]))

plt.xticks(y\_pos, df1["words in Ham"])

plt.title('Repeated words in Ham messages')

plt.xlabel('words')

plt.ylabel('number')

plt.show()

df2.plot.bar(legend = False, color = 'red')

y\_pos = np.arange(len(df2["words in spam"]))

plt.xticks(y\_pos, df2["words in spam"])

plt.title('Repeated words in Spam messages')

plt.xlabel('words')

plt.ylabel('number')

plt.show()

f = feature\_extraction.text.CountVectorizer(stop\_words = 'english')

X = f.fit\_transform(data["v2"])

np.shape(X)

data["v1"]=data["v1"].map({'spam':1,'ham':0})

X\_train, X\_test, y\_train, y\_test = model\_selection.train\_test\_split(X, data['v1'], test\_size=0.33, random\_state=42)

list\_alpha = np.arange(1/100000, 20, 0.11)

score\_train = np.zeros(len(list\_alpha))

score\_test = np.zeros(len(list\_alpha))

recall\_test = np.zeros(len(list\_alpha))

precision\_test= np.zeros(len(list\_alpha))

count = 0

for alpha in list\_alpha:

bayes = naive\_bayes.MultinomialNB(alpha=alpha)

bayes.fit(X\_train, y\_train)

score\_train[count] = bayes.score(X\_train, y\_train)

score\_test[count]= bayes.score(X\_test, y\_test)

recall\_test[count] = metrics.recall\_score(y\_test, bayes.predict(X\_test))

precision\_test[count] = metrics.precision\_score(y\_test, bayes.predict(X\_test))

count = count + 1

matrix = np.matrix(np.c\_[list\_alpha, score\_train, score\_test, recall\_test, precision\_test])

models = pd.DataFrame(data = matrix, columns =

['alpha', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precision'])

models.head(n=10)

best\_index = models['Test Precision'].idxmax()

models.iloc[best\_index, :]

models[models['Test Precision']==1].head(n=5)

best\_index = models[models['Test Precision']==1]['Test Accuracy'].idxmax()

bayes = naive\_bayes.MultinomialNB(alpha=list\_alpha[best\_index])

bayes.fit(X\_train, y\_train)

models.iloc[best\_index, :]

m\_confusion\_test = metrics.confusion\_matrix(y\_test, bayes.predict(X\_test))

pd.DataFrame(data = m\_confusion\_test, columns = ['Predicted 0', 'Predicted 1'],

index = ['Actual 0', 'Actual 1'])

print('confusion matrix for naive bayes\n',m\_confusion\_test)

print(precision\_test)

print(recall\_test)

###############################SVM#############################################

list\_C = np.arange(500, 2000, 100) #100000

score\_train = np.zeros(len(list\_C))

score\_test = np.zeros(len(list\_C))

recall\_test = np.zeros(len(list\_C))

precision\_test= np.zeros(len(list\_C))

count = 0

for C in list\_C:

svc = svm.SVC(C=C)

svc.fit(X\_train, y\_train)

score\_train[count] = svc.score(X\_train, y\_train)

score\_test[count]= svc.score(X\_test, y\_test)

recall\_test[count] = metrics.recall\_score(y\_test, svc.predict(X\_test))

precision\_test[count] = metrics.precision\_score(y\_test, svc.predict(X\_test))

count = count + 1

matrix = np.matrix(np.c\_[list\_C, score\_train, score\_test, recall\_test, precision\_test])

models = pd.DataFrame(data = matrix, columns =

['C', 'Train Accuracy', 'Test Accuracy', 'Test Recall', 'Test Precision'])

models.head(n=10)

best\_index = models['Test Precision'].idxmax()

models.iloc[best\_index, :]

#models[models['Test Precision']==1].head(n=5)

#best\_index = models[models['Test Precision']==1]['Test Accuracy'].idxmax()

svc = svm.SVC(C=list\_C[best\_index])

svc.fit(X\_train, y\_train)

models.iloc[best\_index, :]

m\_confusion\_test = metrics.confusion\_matrix(y\_test, svc.predict(X\_test))

pd.DataFrame(data = m\_confusion\_test, columns = ['Predicted 0', 'Predicted 1'],

index = ['Actual 0', 'Actual 1'])

print('confusion matrix for SVM\n',m\_confusion\_test)

print(precision\_test)

print(recall\_test)

**RESULTS AND OUTPUT:**

The following metrics are used to measure a classifier's performance:

Accuracy: % of predictions that were correct

Recall: % of spam messages that were predicted correctly

Precision: % of messages classified as spam that were spam

Confusion matrix with naïve bayes classifier

|  |  |
| --- | --- |
| Alpha | 15.730010 |
| Train Accuracy | 0.979641 |
| Test Accuracy | 0.969549 |
| Test Recall | 0.777778 |
| Test Precision | 1.000000 |

**A picture containing large, standing, computer

Description automatically generated**

Confusion matrix with SVM classifier

|  |  |
| --- | --- |
| C | 500.000000 |
| Train Accuracy | 0.994910 |
| Test Accuracy | 0.982599 |
| Test Recall | 0.873016 |
| Test Precision | 1.000000 |

**A screenshot of text

Description automatically generated**

The Accuracy for SVM is High compared to Naïve-bayes, So the best model for this project is SVM.

**References**:

[1] http://www.ijstr.org/final-print/nov2019/Detecting-Spam- Emailssms-Using-Naive-Bayes-And-Support-Vector-Machine-.pdf

[2] SMS Spam Collection Data Set from UCI Machine LearningRepository,http://archive.ics.uci.edu/ml/datasets/SMS+Spam+Collection

[3] T. G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, Multiple Classifier Systems, pages 1-15. LNCS Vol. 1857, Springer, 2001.

[4] "Spam Protection Technologies." Securelist.com. Secure list, n.d.Web.30Apr.2014. <http://www.securelist.com/en/threats/spam?chapter=97>.

\*\*\*\*\*\*\*THANK YOU\*\*\*\*\*\*\*