

PROFESSIONAL TRAINING REPORT

At

Sathyabama Institute of Science and Technology

(Deemed To be University)

Submitted in partial fulfillment of the requirements for the award of
Bachelor of Engineering Degree in Computer Science and Engineering

BY

NAME: ROUTHU ASHISH

REGNO: 38110051

NAME: DOMMATI UMESH KUMAR

REGNO: 38110138



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SCHOOL OF COMPUTING

SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

JEPPIAR NAGAR, RAJIV GANDHI SALAI,

CHENNAI-600119, TAMILNADU



SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade "A" by NAAC

(Established Under Section 3 Of UGC Act 1956)

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119

www.sathyabamauniversity.ac.in



DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the Bonafide work of **ROUTHU ASHISH (Reg.no. 38110051)** and **Dommati umesh kumar (Reg.no.38110138)** and who carried out the project entitled "**SPAMCHECKER**" under our supervision from June 2020 to August 2020.

Internal Guide

Dr. N. SRINIVASAN M.E., Ph.D.

Head of the Department

Dr.S. VIGNESHWARI M.E., Ph.D.,

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

I ESLAVATH SAHITHI (Reg.No.37110658) hereby declare that the Project Report Entitled "SPAM CHECKER" done by me under the guidance of Dr .G.NAGARAJAN M.E., Ph.D. at ASPERVISION TECH EDUCATION is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering / Technology degree in Computer Science and Engineering

DATE:

PLACE:

SIGNATURE OF THE CANDIDATE:

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management of **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey thanks to **Dr. Sasikala M.E., Ph.D., Dean**, school of computing and **Dr. S. Vigneshwari, M.E., Ph.D., Head of the Department**, Department of Computer Science Engineering for providing us the necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **Dr. N. Srinivasan M.E., Ph.D.**, for his valuable guidance, suggestions, and constant encouragement paved way for the successful completion of my project.

I wish to express our thanks to all Teaching and Non-teaching staff members of the Department of **COMPUTER SCIENCE AND ENGINEERING** who were helpful in many ways for the completion of the project.

ABSTRACT

Spam Checker Analysis systems are basically binary class or multi-class classification system that classifies feedback of customers into various Specific classes. It has become an industry on its own. There are dozens of notable internet companies, which can be referred as app companies who are doing customer feedback analysis for other, often much larger companies. Many feedback companies like Freshdesk and Nebula do analysis to various internet age companies like Amazon , Google ,Microsoft . Multi-class classification of texts is the challenging task in the field of Machine Learning. Our work focus at the task of six class classification of feedbacks received from different customers. Our corpus is categorized into six class classification namely comment, request, bug, complaint, meaningless and undetermined. The training and testing sets are generated using 10-fold Cross-validation method. We have achieved an accuracy of 64.70% using Random Forest algorithm. However, he baselines accuracy achieved by us is 53.42% using Gaussian Naïve Bayes algorithm.

Contents

ABSTRACT.....	6
CHAPTER 1	11
SPAM CHECKER ANALYSIS	11
CHAPTER 2	13
RELATED WORK.....	13
CHAPTER 3	16
IMPLEMENTATION	16
CHAPTER 4	24
OBSERVATIONS.....	24
CHAPTER 5	27
SUMMARY AND CONCLUSION.....	27
REFERENCES	30

LIST OF FIGURES

FIGURE NO	FIGURE NAME	PAGE NO
1.1	PIE CHART OF DATA DISTRIBUTION	10
1.2	CONFUSION MATRIX	19

LIST OF TABLES

TABLE NO	TABLE NAME	PAGE NO
2.1	CLASS DISTRIBUTION	8
2.2	OBSERVATION	18

LIST OF ABBREVIATIONS

ABBREVIATION	EXPANSION
API	APPLICATION PROGRAM INTERFACE
GUI	GRAPHICAL USER INTERFACE
NB	NAÏVE BAYES
NLP	NATURAL LANGUAGE PROCESSING
SVM	SUPPORT VECTOR MACHINE

CHAPTER 1

SPAM CHECKER ANALYSIS

Introduction:

Spam Checker is a marketing term that describes the process of obtaining a spam opinion about a business, product or service. Spam checker is so important because it provides marketers and business owners with insight that they can use to improve their business, products and/or overall spam experience. Spam analysis measures how happy spam are with a company's products and services. Feedback analysis provides companies with feedback about everything from products to the buying process to support. Most organizations combine this powerful data with other forms of spam to create actionable intelligence about the entire spam journey. Spam is the one thing that gives a business a clearer view of how it is doing. Proper analysis provides a business with a better view of what it has to change, what it has to improve on, and what it has to do, to retain and grow revenue and profit.

Each customer review counts. But it is practically impossible to go through each customer manually. Hence a robust and an efficient system is necessary for the classification of the spam checker of a commodity. We aim to classify a feedback into six classes, namely comment, request, bug, complaint, meaningless and undetermined (further discussion has been done in chapter 4). Using past data we train our system to learn and hence use that acquired knowledge to assign a new spam checker into its suitable class. The whole process makes the evaluation of a product in the future market easy

This thesis consists of 6 chapters. chapter 1 introduces us to the problem statement while chapter 2 elaborates the related work done in this field. chapter 3 discusses the tools that we have used in the project. chapter 4 deals with the implementation of our work in detail. chapter 5 displays the observations obtained and consequently.

1.1. Motivation

One of the main reasons we decided to work together was our passion for natural language processing (NLP) . While the domain of NLP has various sub-domains, we were interested in putting our efforts on the multi-class classification field. So we embarked on a journey to achieve the highest possible accuracy in a spam checker analysis system. The theme of the project was inspired by the fact that it is not possible for companies to go through each of the spam checker for their products manually and decide what should be their priority. Instead, we designed a system which will classify the customer reviews to appropriate classes efficiently.

1.2. Problem statement

As mentioned in the introduction, the objective of the project is to analyze and classify a spam checker into suitable classes. Basic idea is to build an intelligent multi-class classification system.

The most challenging parts in Spam Analysis are:

- It is important that the feedback tool provides us with analysis and automated flows, or else, we could be stuck patch-working different solutions to run one simple spam checker process or worse, do all that manually.
- Another spam checker challenge comes in the final, and arguably, most important step of any checker process. And one can guess, any manual processes will delay the steps in spam checker cycles. It is important to act upon the results of your checker analysis. This means acting upon the good and the bad.
- Most often, because the process requires manual work, companies tend to pay less attention to the good ham and just focus on acting to remedy the bad spam. While this may be enough to get by, it is not the best practice.

CHAPTER 2

RELATED WORK

There has not been a lot of work done in the field of spam checker analysis. However, a lot of research work has been done on multi-class classification.

One other work done is precision of multi-class classification methods for Support Vector Machines by Hongjian et. al [2]. They have used multi-support vector machines together.

They have experimented using one-against-one SVM, one-against-the-rest SVM and binary tree SVM. They have drawn that precision of Binary Tree SVM is better than that of one-against-the-rest SVM, and in all types of Binary Tree SVM, precision of Best Balanced Binary Tree SVM is best, precision of Worst Balanced Binary Tree SVM is worst.

Pingpeng et. al has presented their work MSVM-kNN: Combining SVM and kNN for Multiclass Text Classification [3]. According to their paper, SVM can not identify categories of documents correctly when the texts are in cross zones of multi-categories, kNN cannot effectively solve the problem of overlapped categories borders. However, their experiments have shown that using SVM for identifying category borders, and then using kNN for classifying documents among the borders improve performance. Their experimental results show that their approach Multi-class SVM-kNN (MSVM-kNN) performs better than SVM or kNN.

Gamon, M. [4] describes in his paper on sentiment classification on spam checker data that large feature vectors in combination with feature reduction can train linear support vector machine that can achieve high classification accuracy. The paper suggests that the addition of deep linguistic analysis features to a set of surface level word n-gram features contributes consistently to classification accuracy in this domain. Another research by Pal, M. on Random forest classifier for remote sensing classification [5] shows that random forest classifier performs equally well to

SVMs in terms of classification accuracy and training time. Another research, an improved random forest classifier for multi-class classification by Chaudhary et. al [7] showed a greater increase in disease classification accuracy as 97.80% as compared to Neural Networks, Logistic Regression and SVM as 92.20%, 94.80%, 95.70% respectively.

2.1 IDE & Text Editors

An Integrated Development Environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and debugger.

2.1.1 Python Note book

Python is a command shell for interactive computing in multiple programming languages, originally developed for the Python programming language, that offers introspection, rich media, shell syntax, tab completion, and history. Python provides the following features:

- Interactive shells (terminal and Qt-based).
- A browser-based notebook with support for code, text, mathematical expressions, inline plots and other media.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into one's own projects.

2.2 Languages & Technologies

2.2.1 Python:

Python is a widely used high-level, general-purpose, interpreted, dynamic programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than possible in languages such as C++ or Java. The language provides constructs intended to enable writing clear programs on both a small and large scale.

2.3 Python Libraries

2.3.1 SK Learn:

It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, k-means and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

2.3.2 Pandas:

Pandas is an extension to the Python programming language, adding support for data Manipulation and analysis. In particular it offers data structures and operations for manipulating numerical tables and time series.

2.3.3 Numpy:

NumPy is an extension to the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays. The ancestor of NumPy, Numeric, was originally created.

2.3.4 Gensim:

Genism is a robust open-source vector space modeling and topic modeling toolkit implemented in Python. It uses NumPy, SciPy and optionally Python for performance. Genism is specifically designed to handle large text collections, using data streaming and efficient incremental algorithms, which differentiates it from most other scientific software packages that only target batch and in memory processing.

2.3.5 Pickle:

Python pickle module is used for serializing and de-serializing a Python object structure.

CHAPTER 3

IMPLEMENTATION

3.1 Introduction

In Implementation part we have combined all the things we had done in earlier phases of our project development process. We have used all the technologies described in requirement analysis part to achieve all the functionalities of our system. We used supervised learning approach of machine training approach to train our system to achieve better classification.

Our approach of implementing Spam Analysis System was divided into 4 parts, namely:

- Corpus Collection
- Feature Extraction
- Splitting of Dataset
- Training and Modeling

3.1.1. Corpus Collection

In a joint ADAPT (ADAPT Venture)-Microsoft research project, representative real world samples of customers' feed-backs has been released jointly by Microsoft and IJCNLP (International Joint Conference on Natural Language Processing).

About the corpus

Syntax:

```
dictionary = gensim.corpora.Dictionary(mails)
```

corpora means collection of data i.e, mails

Number of documents = 3565

Number of tokens = 45135

Number of classes = 6

Class	No. of documents	No. of tokens
Bug	88	1553

Comment	1921	22099
Complaint	1063	15720
Meaningless	354	3600
Request	114	1827
Undefined	25	336
Total	3565	45135

Table 2.1: Class distribution

In our corpus, there are a total no. of 3565 documents, which are distributed into six classes. From them, comment & complaint have maximum no. of documents, numerically 1921 and 1063 respectively. Other classes such as Bug, Meaningless & Request have 88, 354 & 114 documents respectively.

- Comment:

A comment is a checker that is basically a remark, an opinion or simply an observation. e.g.

- 1) Basically, it's an ok planner.
- 2) It is so wonderful to use.

- Request:

A request can be a plea or an application. e.g.

- 1) Being a new Apple Developer, I needed a fast program that would work fast and has a easy User Interface.
- 2) Even if they are planning to renovate the old section in the near future, it would be nice if they kept up with simple maintenance issues - like replacing the missing tiles from the stairs etc...

- Bug:

A bug is a spam telling that the product is flawed or it is erroneous. e.g.

- 1) Plus, lots of graphical lag when trying to show new messages.

2) Phone froze as if the app had a virus.

- Meaningless:

This refers to the irrelevant feed-backs that has nothing to do with the particular product. e.g.

1) Beautiful afternoon at the Bristol!

2) Need to know which all programming languages I need to know to start working on Android SDK?

- Complaint:

This is the type of spam denotes the customers' dissatisfaction of the product, or the customer wants some specific things to be done. e.g.

1) Even the accessories in the app look fake.

3) I don't what Microsoft was thinking with this update but I don't like it either way.

- Undetermined:

These are those kind of feed-backs that is not only irrelevant but also can not be interpreted. e.g.

1) Unless one of them DIES (rip lightbulb you will be missed.

2) Maybe old style clothing too from civil war era not just city slicker clothing.

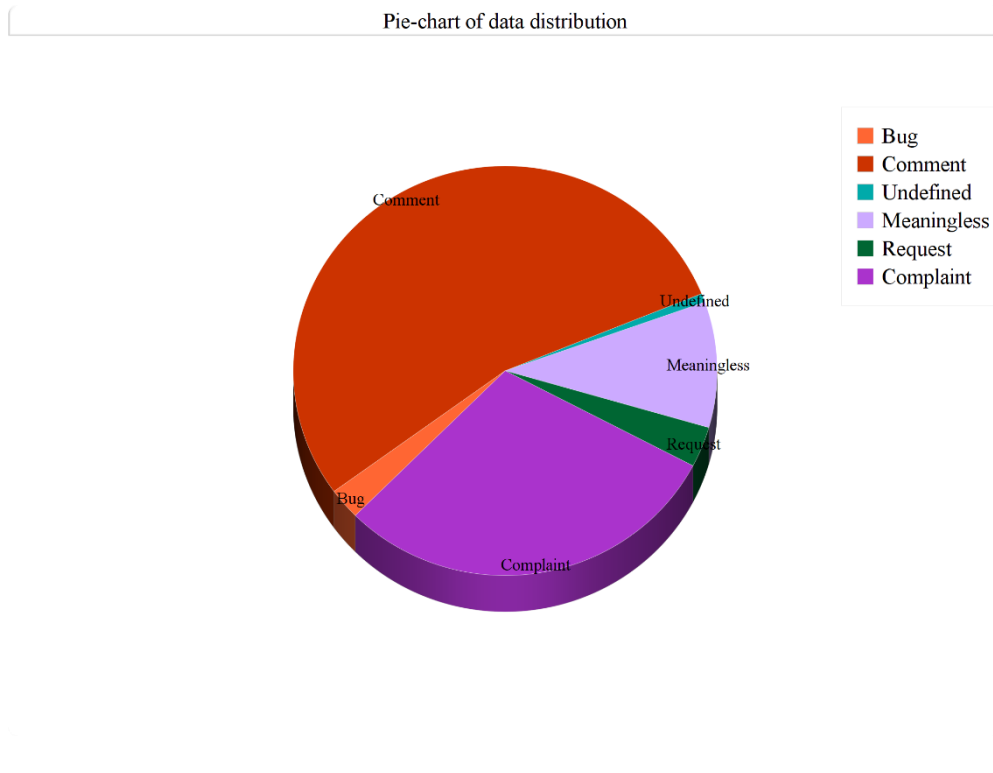


Figure 3.1 : Pie-chart of data distribution

Tokenizing each document:

Tokenizing [17] : Next step, after pre-processing the data we tokenize the documents. It is an important pre-processing step. For example:

Wow!! I'm in love with Windows10

The tokenization of the above section will be

Wow!! I am in love with Windows10

We have used NLTK sentence tokenizer for tokenizing sentences and NLTK word tokenizer for tokenizing words.

3.1.2 Feature Extraction

Transforming the corpus into vector space using TF-IDF and Bag-of-words .After getting the refined data, we started converting the corpus to vector space using TF-IDF features and Bag-of-Words. We made a dictionary from the words of the documents.

Bag-of-words

The bag-of-words model is a simplifying representation used in natural language processing and information retrieval . In this model, a text (such as a sentence or a document) is represented as the bag (multiset) of its words, disregarding grammar and even word order but keeping multiplicity. The bag-of-words model has also been used for computer vision.

SYNTAX:

```
bag_of_words = [ dictionary.doc2bow(mail) for mail in mails]
dense_bow     = gensim.matutils.corpus2dense(bag_of_words, num_terms =
n_unique_tokens).transpose()
```

BOW- BAG OF WORDS: To converts list of words into list of numbers.

Train-set=dense-bow[:values]

Test-set=dense-bow[values:

The TF-IDF weight is composed by the two terms: Term Frequency, and Inverse document frequency.

- Term Frequency:

Which measures how frequently a term occurs in a document. Since every document is different in length, it is possible that a term would appear much more times in long documents than shorter ones. Thus, the term frequency is often divided by the document length (aka. the total number of terms in the document) as a way of normalization:

$$TF(word) = Fcount(word)/N$$

Fcount(word): number of times "word" is found in a documents

N: Total number of terms in a document

- Inverse Document Frequency:

Which measures how important a term is. While computing TF, all terms are considered equally important. However it is known that certain terms, such as "is", "of", and "that", may appear a lot of times but have little importance. Thus we need to weigh down the frequent terms while scale up the rare ones, by computing the following:

$$IDF(word) = \log_e(N/Ecount(word))$$

Ecount(word): number of documents with term “word”

N: Total number of terms in a document

$\text{tfidf}(\text{word}) = \text{TF}(\text{word}) * \text{IDF}(\text{words})$

3.1.3 Splitting of Dataset

After making the vector space model we divided it into two parts. First part is used for training our model and the other one is used to test our machine so we divided the entire dataset in the ratio of 9:1, 90% for training and 10% for testing purpose.

The main reason is that the amount of data available is limited. We have done 10-fold cross validation which reduces the variance by averaging over 10 different partitions. So, the performance estimate is less sensitive to the portioning of the data.

K-FOLD ALGORITHM

In k-fold cross-validation, the original sample is randomly partitioned into k equal sized subsamples. Of the k sub-samples, a single sub-sample is retained as the validation data for testing the model, and the remaining k - 1 sub-samples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k sub-samples used exactly once as the validation data. The k results from the folds can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling.

That all observations are used for both training and validation, and each observation is used for validation exactly once. 10-fold cross-validation is commonly used, but in general k remains an unfixed parameter. For example, setting k = 2 results in 2-fold cross-validation.

In 2-fold cross-validation, we randomly shuffle the dataset into two sets d0 and d1, so that both sets are equal size (this is usually implemented by shuffling the data array and then splitting it in two). We then train on d0 and validate on d1, followed by training on d1 and validating on d0. When k = n (the number of observations), the k-fold cross-validation is exactly the leave-one-out cross validation. Using k-fold algorithm we have divided the dataset into 10 bins. The algorithm is run 10 times and every time 9 bins are used for training and 1 bin is used for testing.

SYNTAX:

```
from sklearn.model_selection import KFold
```

```
kf = KFold(n_splits=10,shuffle=True)
```

3.1.4 TRAINING AND MODELING

Applying training Algorithm

After splitting the training and testing data, we started training our model with different classifier for a comparative observation.

First we implemented the Gaussian Naive Bayes classifier.

We took the training data set along with their output labels and trained our model with Gaussian Naive Bayes classifier. Then we tested the model on the testing dataset and achieved an accuracy of 53.42%.

After Implementation of Gaussian Naive Bayes classifier we tried Multinomial Naive Bayes classifier.

Multinomial Naive Bayes classifier. Then we tested the model on the testing dataset and achieved an accuracy of 55.49%.

After this we have tried Bernoulli Naive Bayes classifier for training the model.

Bernoulli naive Bayes Classifier: Like MultinomialNB, this classifier is suitable for discrete data. The difference is that while MultinomialNB works with occurrence counts, BernoulliNB [16] is designed for binary/Boolean features.

```
for train, test in kf.split(dataset):  
    train_set.append(dataset[i])  
    train_labels.append(target[i])  
    test_set.append(dataset[i])  
    test_labels.append(target[i])
```

We trained our model with the training data set along with their output labels with Bernoulli Naive Bayes classifier. Then we tested the model on the testing dataset and achieved an accuracy of 55.09%.

After Naive Bayes classifiers we tried K nearest neighbor algorithm for training purpose.

- In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.
- In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

We did training of model with k nearest neighbor where k is 4. and did our testing with help of testing Dataset and achieved increment in accuracy so accuracy of 57.46% was achieved. After k-NN classifier we used Support Vector Machine for training purpose and tested the data.

- Support Vector Machine:

In machine learning, support vector machines [13] are supervised learning models with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, a SVM training algorithm builds a model that assigns new examples to one category Making it a non-probabilistic binary linear classifier.

SYNTAX:

from sklearn import svm

SVM are used to supervised learning methods for classification & clustering.

CHAPTER 4

OBSERVATIONS

After applying all the classifiers we held comparative observations among the classifiers. We have compared them via the factors of Accuracy, Recall value, F1-score (Combination of precision Recall values) .

The below comparing measures are used to compare different approaches that were used in spam checker analysis model.

4.1 ACCURACY

The degree to which the result of a measurement, calculation, or specification conforms to a correct of a standard.

$$\text{Accuracy} = \frac{\text{number of correct Feedback} \times 100}{\text{Total No. of Feedback}}$$

4.2 F1 Scores

The F-beta score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

F1 Score - Micro: Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

F1 Score - Macro: Calculate metrics globally by counting the total true positives, false negatives and false positives.

F1 Score - Weighted: Calculate metrics for each label, and find their average, weighted by support (the number of true instances for each label).

$$\text{F1Score} = \frac{2 \times \text{precision} \times \text{recall}}{(\text{precision} + \text{recall})}$$

We performed error analysis with the help of confusion matrix.

Classifier	Accuracy	Precision Score	Recall Score	F1 Score
Gaussian Naive Bayes	53.42	0.53	0.53	0.53

Multinomial Naive Bayes	55.59	0.56	0.56	0.56
Bernoulli Naive Bayes	55.09	0.55	0.55	0.55
Support-Vector Machine	58.50	0.59	0.59	0.59

Table 2.2: Observation table

Confusion matrix:

A confusion matrix is also known as error matrix is a specific table layout that allows visualization of the performance of an algorithm, typically a supervised learning one. Each column of the matrix represents the instances in a predicted class while each row represents the instances in a each class

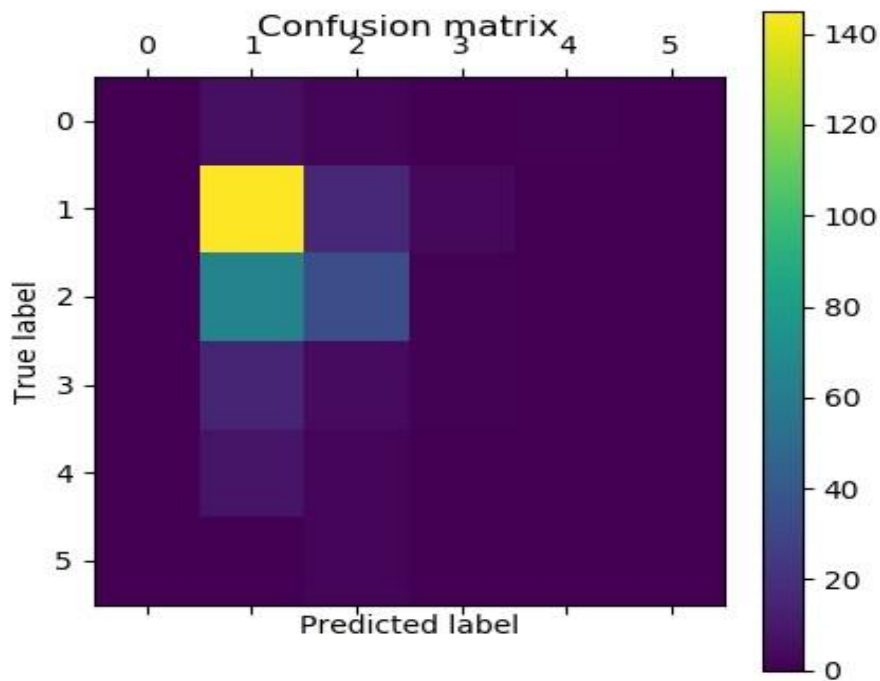


Figure 1.2 of confusion matrix

Random Forest Confusion Matrix From the Figure 1.2 of confusion matrix, we infer that maximum errors were found in differentiating between comment and complaint classes. After this, most errors were found in differentiating between meaningless and undefined.

CHAPTER 5

SUMMARY AND CONCLUSION

5.1 APPENDIX

SCREEN

```
a) import numpy as np
import pandas as pd
df = pd.read.csv('spam.csv',encoding='latin-1')
df
stopwords = []
def get_the_sw():
    with open('stopwords.txt') as stops:
        for i in stops:
            stopwords.append(i.strip())
get_the_sw()
stopwords
mails = []
def get_the_corpus():
    for i in df.v2:
        mail = i.strip().split()
        mail = [word for word in mail if word not in stopwords ]
        mails.append(mail)
get_the_corpus()
mails
from collections import defaultdict
frequency = defaultdict(int)
def removal_signal_freq():
    for mail in mails:
        for word in mail:
            frequency[word] +=1
removal_signal_freq()
mails = [[word for word in mail if frequency[word]>1 for mail in mails]
mails
import gensim
dictionary = gensim.corpora.Dictionary(mails)
bag_of_words = [dictionary.doc2bow(mail) for mail in mails]
n_unique_tokens = len(dictionary)

dense_bow=gensim.matutils.corpus2dense(bag_of_words,num_terms=
n_unique_tokens).transpose()
dataset = dense_bow
target =[]
for i in df.v1:
```

```

    if i == 'ham':
        target.append(1)

    elif i == 'spam'
        target.append(0)
target
import pickle
filename = 'data_vec.pkl'
file_vec = open(filename,'wb')
pickle.dump(dataset,file_vec)
file_vec.close()
file_vec = open(filename,'rb')
dataset = pickle.load(file_vec)
file_vec.close()
dataset
from sklearn.model_selection import KFold
    kf = KFold(n_splits=10,shuffle=True)
from sklearn import svm\n",
    from sklearn import naive_bayes as nb
    from sklearn.metrics import f1_score
    from numpy import mean
    accuracies = []
    scores = []
        for train, test in kf.split(dataset):\n",
            print("\n1\n")
            train_set = []
            train_labels = []
            test_set = []
            test_labels = []
            for i in train:
                train_set.append(dataset[i])
                train_labels.append(target[i])
            for i in test:
                test_set.append(dataset[i])
                test_labels.append(target[i])
#    classifier = nb.GaussianNB()
    classifier = svm.SVC()
    predicted = classifier.fit(train_set, train_labels).predict(test_set)
    score = f1_score(test_labels, predicted, average = 'weighted')

scores.append(score)
    incorrect = (test_labels != predicted).sum()
    accuracy = (len(test_set) - incorrect) / len(test_set) * 100
    accuracies.append(accuracy)
#    print("\n1\n")
print("\nMaximum accuracy attained \", max(accuracies))

```

```
print('f1score ', scores[np.argmax(accuracies)])  
    print('\n')  
accuracies  
max(accuracies)
```

5.2 Conclusion And Future Scope

Working on multi-class classification that too for six classification of raw comments is not an easy task in the field of NLP & Machine Learning. After preprocessing of corpus, we employed 10-fold cross-validation method for training and testing purpose. We employed various machine learning algorithms to get the best model.

Initially we achieved baseline accuracy of 53.42% using Gaussian Naive Bayes algorithm. Finally we got an accuracy of 64.70% using Random forest, followed by accuracy of 55.59% using Multinomial Naive Byes, 55.09% using Bernoulli Naive Bayes, 58.49% using SVM and 57.65% using KNN classifiers respectively.

REFERENCES

- 1) Li, H., Jiao, R., & Fan, J. (2008, October). Precision of multi-class classification methods for support vector machines. In Signal Processing, 2008. ICSP 2008. 9th International Conference on (pp. 1516-1519). IEEE.
- 2) Yuan, P., Chen, Y., Jin, H., & Huang, L. (2008, July). MSVM-kNN: Combining SVM and k-NN for Multi-class Text Classification. In Semantic Computing and Systems, 2008.WSCS'08. IEEE International Workshop on (pp. 133-140). IEEE. Porter, M. F. (2001). Snowball: A language for stemming algorithms.[19].
- 3) Rifkin, R., & Klautau, A. (2016). In defense of one-vs-all classification. Journal of machine learning research, 5(Jan), 101-141.
- 4) Popescu, M. (2017). Regularized least-squares classification for word sense disambiguation sign (f (x), 1, 1.
- 5) Read, J., & Perez-Cruz, F. (2018). Deep learning for multi-label classification. ArXiv preprint arXiv:1502.05988.