SPAM EMAIL DETECTION WITH BIO-OPTIMIZED MACHINE LEARNING ALGORITHMS

Report submitted to the SASTRA Deemed to be University as the requirement for the course

CSE300 - MINI PROJECT

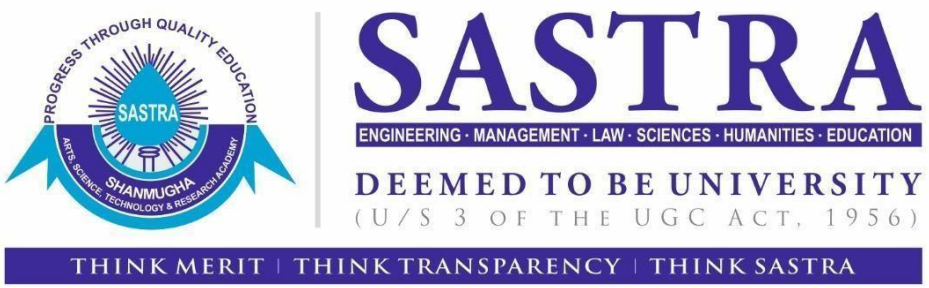
Submitted by

ARUNA SRI S
(123003024, B. TECH Computer Science and Engineering)

PAVITRA R
(123003185, B. TECH Computer Science and Engineering)

June 2022

SCHOOL OF COMPUTING
THANJAVUR, TAMIL NADU, INDIA – 613 40



SCHOOL OF COMPUTING THANJAVUR– 613 401

Bonafide Certificate

This is to certify that the report titled "**Spam email detection with bio-optimized machine learning algorithms**" submitted as a requirement for the course, CSE300: **MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Ms. Aruna Sri S (123003024, B. Tech Computer Science and Engineering**), **Ms. Pavitra R (123003185, B. Tech Computer Science and Engineering),** during the academic year 2021-22, in the School of Computing, under my supervision.

Signature of Project Supervisor :

Name with Affiliation : Dr. P. Saravanan, Asst.Professor-III – CSE,
School of Computing, SASTRA Deemed University

Date :

Mini Project *Viva voce* held on 25.06.2022

Examiner I **Examiner II**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan,** Dean**,** Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks to **Dr. R. Chandramouli,** Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology and Information & Communication Technology and **Dr. B. Santhi**, Associate Dean, Department of Computer Application.

Our guide **Dr. P. Saravanan**, Asst.Professor -III - CSE, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. We thank you all for providing me an opportunity to showcase my skills through this project.

List of figures

Figure No.	Figure Name	Page No
Fig 4.1	Spam Detector GUI predicting spam email	32
Fig 4.2	Spam Detector GUI predicting ham email	32

List of tables

Table No.	Table name	Page No.
Table 2.4.1	Performance Measure- Ling Spam Dataset	10
Table 2.4.2	Performance Measure- Spam Assassin Dataset	10
Table 2.4.3	Performance Measure- Enron Dataset	10
Table 2.4.4	Performance Measure- PU1 Dataset	11
Table 2.4.5	Performance Measure- PU2 Dataset	11
Table 2.4.6	Performance Measure- PU3 Dataset	11
Table 2.4.7	Performance Measure- PUA Dataset	12
Table 2.4.8	Accuracy after optimizing the base models	12

Abbreviation

MNB	Multinomial Naive Bayes
SVM	Support Vector Machine
SGD	Stochastic Gradient Descent
DT	Decision Tree
RF	Random Forest
MLP	Multilayer Perceptron
PSO	Particle Swarm Optimization
GA	Genetic Algorithm
ANN	Artificial Neural Network
ML	Machine Learning
NN	Neural Network
TN	True Negative
TP	True Positive
FP	False Positive
FN	False Negative
NB	Naive Bayes
K-NN	K-Nearest Neighbor
LR	Logistic Regression
ISP	Internet Service Provider
BSO	Bee Swarm Optimization
PCA	Principal Component Analysis
ACO	Ant Colony Optimization
ECS	Enhanced Cuckoo Search
GUI	Graphical User Interface

Abstract

In today's fast-paced world, information transfer methods such as electronic mails have made communication seamless and instantaneous. Spammers use this as an advantage to make fraudulent gains by sending unbidden mails. This creates inconvenience be it individuals or any organizations. The primary goal of our paper is to exploit bio-inspired machine learning algorithms to detect spam emails. A survey is carried out to traverse through the efficient methodologies used on various datasets to gain good results. An ample amount of research was done to execute machine learning models utilizing Support Vector Machine, Decision Tree, Naive Bayes, Random Forest and Multi-Layer Perceptron on seven various email corpus alongside feature extraction and pre-processing. We employed bio-inspired methods like Genetic Algorithm and Particle Swarm Optimization to optimize the classifier's performance. Among the rest, Multinomial Naive Bayes with Genetic Algorithm produced best results. Our findings are compared to those of other machine learning and bio-inspired models in order to choose the most appropriate model.

KEY WORDS: Electronic mail(email), Machine learning (ML), Bio-inspired algorithms, Particle Swarm Optimization, Genetic Algorithm

Table of Contents

Title	Page No.
Bonafide Certificate	ii
Acknowledgements	iii
List of Figures	iv
List of Tables	v
Abbreviation	vi
Abstract	vii
1. Summary of the base paper	1
2. Merits and Demerits of the base paper	7
3. Source Code	13
4. Snapshots	32
5. Conclusion and Future Plans	33
6. References	34
7. Appendix – Base Paper	36

CHAPTER 1

SUMMARY OF THE BASE PAPER

S. Gibson, B. Issac, L. Zhang and S. M. Jacob, "Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms," in IEEE Access, vol. 8, pp. 187914-187932,2020, doi: 10.1109/ACCESS.2020.3030751, Scopus, SCI Indexed

1.1 Introduction

Machine learning techniques are applied in various fields of computer science from product recommendation to disease prediction. Emails are one of the fastest and most popular modes of internet communication used for both business and personal conversations, also communicating through an email is one of the affordable and easy ways of usage for official purposes. In recent years, unsolicited illegitimate emails are sent in huge amounts to random recipients by spammers. The bulk volume of spam emails has devastating effects on the time, memory, power and communication bandwidth. These emails may contain spam links that lead to phishing attacks, or contain malware to get unauthorized access to the recipient's device. The spammers may even have access to the user's personal details including credit cards and bank accounts. They further use certain complex unethical techniques to hinder the detection of spam. Thus, there is a huge demand for techniques to be developed to filter out spam emails from the legitimate ones and to avoid the spammers from further sending spam mails.

Spam detection in networks is done using various tools and techniques that are offered by companies. Spam detection is an approach based on text mining, where the available email content is classified as spam or ham. Researchers have worked and are continuously working on various machine learning models and computational intelligence-based methods to detect and classify emails. Filtering techniques have been set up to detect unsolicited emails by setting up rules and establishing firewall settings. Google offers spam detection and filtering machine learning models having 99.9 percent accuracy. Spam filters can be deployed in different areas such as on a user's computer, cloud hosted applications or on the routers. Content-based filtering, heuristic or rule-based filtering, case-based spam filtering, the previous likeness-based spam filtering, adaptive spam filtering and Bayesian filtering have all been used to solve the problem of spam email detection.

In knowledge engineering, spam detection rules are established, and spam detection is based on knowledge base mining, mail header analysis and cross validation. Since they require constant manual updating, it is time and resource consuming. Machine learning on the other hand makes the task easier as it learns to categorize spam and ham emails and applies the acquired instructions to the unknown emails.

The spam classification problem can further be assessed by automated parameter selection or feature selection for the models. For this study, the publicly available datasets that contained each email as text files were collected. This paper analyzes the working of five machine learning models on each dataset individually. The models are then optimized with Particle Swarm Optimization and Genetic Algorithm. The optimized results are compared with that of base models to conclude if the models improved the performance with parameter tuning.

1.2 Algorithm

1.2.1 Naïve Bayes - Multinomial

Naive Bayes uses probabilistic techniques to solve the classification problems. It is a supervised learning approach based on Bayes Theorem. The algorithm can be denoted as equation (1):

$$P(\text{Class}|\text{WORD}) = \frac{(P(\text{WORD}|\text{Class}) \times P(\text{Class}))}{(P(\text{WORD}))} \quad (1)$$

where 'WORD' is individual words (word 1, word 2…. word n) from the email and 'Class' is 'Spam' or 'Ham'. Here, P(Class|WORD) is posterior probability, P(Class) is prior probability and P(WORD|Class) is likelihood probability. With 'Class' = 'Spam', the equation - 1 is simplifies as equation (2):

$$P(\text{Class}|\text{WORD}) = \frac{\Gamma_{i=1}^{n} P(\text{word\_i}|\text{Spam}) \times P(\text{Spam})}{P(\text{word\_1, word\_2, \ldots word\_n})} \quad (2)$$

As Multinomial Naive Bayes performs better than Bernoulli and Gaussian, it is selected in this paper for identifying spam emails from text data. This classifier focuses on term frequency and is based on Multinomial Distribution. It can be denoted as in equation (3):

$$P(p|n) \propto P(p) \prod_{1 \le k \le nd} P(t_k|p) \quad (3)$$

where 'nd' is the number of token, 'n' is the number of emails and P (tk |p) - the conditional probability is calculated as in equation (4):

$$P(t_k|p) = \frac{(\text{count}(t_k|p) + 1)}{(\text{count}(t_p) + |V|)} \quad (4)$$

Here, tk is the occurrence of spam term, |V| and 1 are the smoothing constants.

The default values of the parameters for this model are:

Alpha: '1.0', Fit Prior: 'True', Class Prior: 'None'.

1.2.2 Support Vector Machine

SVM is a popular supervised learning method that clusters the data points into classes by a hyperplane. The hyperplane is given in equation (5):

$$H = VX + c \tag{5}$$

where 'V' is the vector and 'C' is a constant. This paper uses an optimized version of SVM (i.e), Stochastic Gradient Descent (SGD) as it provides efficiency while handling large datasets.

The algorithm is implemented with parameters like Alpha, Epsilon and Tol. The optimal learning rate used by the model to iterate over the data is given in in equation (6):

$$\frac{1}{\alpha(t_0 + t)} \tag{6}$$

where 'α' is regularization term, 't0' is heuristic approach and 't' is time step.

1.2.3 Decision Tree Classifier

DT is a supervised learning method, is a tree- structured classifier in which the features are represented by the internal nodes, the decision rules by the branches and the outcome by leaf node. More complicated rules are executed as the tree becomes deeper and longer.

The default parameters used with this model are:

Criterion: 'Gini' or 'Entropy', Splitter: 'best'

The formula for calculating Gini and Entropy is shown in equation (7) and equation (8):

$$\text{Gini} : G_i = 1 - \sum_{k=1}^{n} p_{(i,k)^2} \tag{7}$$

$$\text{Entropy} : H_i = \sum_{\substack{k=1 \\ p_i k \neq 0}}^{n} p_{(i,k)} \log_2(p_{(i,k)}) \tag{8}$$

Where 'i' is the node from training data and 'P' is the probability.

1.2.4 Random Forest Classifier

RF is a supervised learning algorithm based on ensemble learning. It uses multiple decision trees in which each tree predicts the class. The class which is predicted in higher numbers is assigned as the prediction. It is a classifier that uses many decision trees on different subsets of the input dataset and averages the results to increase the dataset's predicted accuracy.

1.2.5 Multi-Layer Perceptron

MLP is a supervised method which is a feed forward ANN. It uses a linear activation function with default value as Hyperbolic Tan. This algorithm uses one or more non-linear layers called hidden layers between the input and the output. These layers accept inputs from the previous layer, and the output layer finally provides the output values. Each hidden layer function is given by Equation (9):

$$f(x) = W_{2g}(W_1^T x + b_1) + b_2 \quad (9)$$

where 'W1' is the weight from the input layer and 'W2' is the weight from the hidden layer.

1.3 Optimization Techniques:

This paper uses two bio- inspired approaches to improve the results obtained with the base models. They are Particle Swarm Optimization (PSO) and Genetic Algorithm (GA).

1.3.1 Particle Swarm Optimization:

The PSO is based on swarming behaviors seen in birds or fish. The overall global position and best position of the particles are used to evaluate them. To determine the overall optimal place, particles are dispersed within a search space. For PSO to be utilized with an ML model, the Pyswarms package provides several calculations and strategies like parameter tuning optimization or feature subset selection. The feature selection process, which has been studied in the preceding sections, has the potential to minimize feature space while also eliminating some features that may be crucial for classification. PSO will therefore be used to fine-tune and identify the hyper-parameter of a certain ML/NN model.

The algorithm uses the objective function - 'optimize' from the 'gobal_best' module to find the 'Global Best Position' and 'Global Best Cost'. The equation of particle position is given as:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

where 'xi' is the particle position, 't+1' is computed velocity and 't' is the current timestamp. The velocity 'vi' is given as:

$$v_{ij}(t+1) = w \times v_{ij}(t) + c_1 r_1 j(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)]$$

here r1 and r2 are random numbers. c2 isa social parameter, c1 is a cognitive parameter, w is the inertia parameter. These parameters are set to their default values. The parameters passed in 'global_best' module is:

Number of particles :10, Dimension, Options: W = 0.9, C1 =0.5 and C2 = 0.7, Bounds

The parameter values (Dimension and Bounds) are passed with respect to the base models. This algorithm takes more computation time than base models.

1.3.2 Genetic Algorithm:

Genetic Algorithm uses Darwinian Natural Selection as its base for selecting the fittest individual in the population sample. It includes the principles like inheritance, variation and selection. Each individual of the population is marked with a unique number that is represented in binary. The algorithm uses the fitness function to find the best individuals for reproducing offspring. Higher fitness rate results in higher probability. The TPOT library is used in implementing GA, it selects the best parameter and then the training and cross validation are done. The parameters passed into TPOT are as follows:

Generation: 10, Population size :40, Offspring size: 20, Mutation rate: 0.9, Crossover rate: 0.1

The values given for the parameters mean that 400 (40*10) combinations of hyperparameters will be evaluated in one generation. With 10- fold cross validation each pipeline is evaluated 4000 times. At termination, the classifier provides the best parameters of the pipeline. The sum of mutation and crossover rates must not exceed 1.

1.4 Performance Measures:

The performance of the base models mentioned above are assessed using various metrics as follows:

1.4.1 Confusion Matrix: It is used to visualize the spam email detection for the base models. It is a matrix with values as follows:

TN: Ham mail is predicted correctly as Ham

TP: Spam mail is predicted correctly as Spam

FP: Spam mail is wrongly predicted as Ham

FN: Ham mail is wrongly predicted as Spam

1.4.2 Accuracy: The ratio of the total number of predictions produced to the number of classifications a model properly predicts is known as model accuracy. The formula is as follows:

$$\text{Accuracy} = (TN+TP)/(TP+FN+FP+TN)$$

1.4.3 Recall: The ratio of correctly predicted spam to total number of spams. The formula for recall is as follows:

$$\text{Recall} = TP/(TP+FN)$$

1.4.4 Precision: The ratio of mails correctly classified as positive to the total number of mails predicted as positive. The formula is given below

$$\text{Precision} = TP/(TP+FP)$$

1.4.5 F1-Score: It is defined as the predictive performance of the model. It is defined as in equation given below:

$$F_\beta = \frac{(1+\beta^2)(\text{precision} \times \text{recall})}{(\beta^2 \times (\text{precision} + \text{recall}))}$$

Here, 'β' is given a value as 1 and the equation is simplified as:

$$F_\beta = \frac{2 \times (\text{precision} \times \text{recall})}{(1 \times \text{precision} + \text{recall})}$$

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 Literature Review

2.1.1 Machine Learning

Machine learning techniques for detecting spam emails have been developed by researchers.

In ref. [1], the authors proposed a support machine vector based naive Bayes - SVM-NB - filtering system, (i.e.) a hybrid system between SVM and NB. The SVM algorithm is used to build the hyperplane between the specified dimensions. The NB algorithm will then be used to predict the likelihood of the result using this set. This SVM-NB hybrid system used the dataset obtained from DATAMALL. Increase in spam-detection accuracy than the individual accuracies of NB and SVM was seen with the SVM-NB model and the classification speed was also faster. The fact that this study only used one dataset (Chinese text corpus) is a major setback.

In ref. [2], the popular machine learning models are reviewed and the experiment was conducted on several algorithms that includes Naïve Bayes (NB) classification, Artificial Neural Network (ANN), Support Vector Machine (SVM), Artificial Immune System and Rough Sets, K-Nearest Neighbor (K-NN). The algorithm descriptions are presented and their performance is compared on the Spam Assassin dataset. Their goal was to mimic human's detection and recognition abilities with tokenization investigated in two stages: training and filtering. The workflow involves dataset preprocessing, feature description, classification of spam and evaluating performance. The Naive Bayes classifier was shown to have the best accuracy, although feature dependency was revealed to be a fundamental flaw in this model.

In ref. [3], the authors of this paper experimented with classifiers such as: K-Nearest Neighbor (K-NN), Naïve Bayes (NB), Support Vector Machine (SVM), Tree and Rule based algorithms. They came up with a vocabulary of ham and spam emails, which they then utilized to filter the training, held-out data. This vocabulary is used to generate labels of training and testing which are used by several machine learning models. NB followed by SVM was found to perform better among the rest. Limited usage of the dataset was a demerit of this work.

In ref. [4], the authors proposed a hybrid system by integrating Logistic Regression (LR) with Decision Tree and False Negative threshold. With the use of LR to reduce noisy data before feeding it to DT, its performance was increased, showing accuracy of 91.67%. Spam Base was the dataset used. It was concluded that LR was able to improve the

efficiency of DT. Noisy data reduction was a challenging task in this proposed method.

In ref. [5], a comprehensive examination of some of the most widely used machine learning-based email spam filtering methods was carried out. The discussion in this study looks at how machine learning techniques are being applied to the email spam filtering processes of major internet service providers (ISPs) like Gmail, Outlook and Yahoo. The review also compares the strengths and weaknesses of each machine learning method. A complete study of this paper is required to get a strong hold on effectively handling spam email classification.

2.1.2 Bio-Inspired Methods

Bi-optimized machine learning algorithms were extensively studied to improve the accuracy in the detection of spam emails.

In ref. [6], an integrated approach comprising the machine learning model- Naive Bayes along with the computational model - Particle Swarm Optimization has been experimented on Ling-Spam corpus. NB is used for learning, classification. PSO is used for optimization of parameters of NB. Correlation Feature Technique was used to select necessary features from the dataset. The performance is measured in terms of precision, recall, f-measure and accuracy. Though feature selection was a tedious process, this method improves the accuracy in detection when compared to NB.

In ref. [7], the authors explored SVM on Arabic text with Bee Swarm Optimization (BSO) and Chi-Squared. The approaches they used are SVM, Neural Networks, SVM optimized BSO and Chi-Squared. BSO is inspired from the behavior of swarm bees, and is used to find global best position. The major problem with this approach is that feature selection is NP -hard and it increases the computation time. The dataset is generated and Chi2-BSO is applied on it to select the features which are then used in SVM. It is also found that SVM optimized with BSO performed better than neural networks. The accuracy of the proposed algorithm was about 95.67%.

In ref. [8], Genetic algorithms are optimized by integration with DT wherein the feature extraction is overcome by Principal Component Analysis (PCA). The J-48 algorithm was used to find the optimal parameter value. The proposed GADT algorithm was experimented on Enron dataset and showed higher accuracy than classifiers without PCA.

In ref. [9], SVM optimized with Ant Colony Optimization (ACO) was experimented on Spam Base dataset. The ants' behavior observed while developing the shortest path to the food source is the basis for ACO. Also, an overview of machine learning algorithms such as KNN, SVM and NB were reviewed in this paper.

It was concluded that the proposed method improved the accuracy of SVM by 4%.
The limitation of this paper is that the proposed algorithm takes more computation time and deducts memory requirement.

In ref. [10], researchers explored various optimization techniques such as FireFly and Cuckoo search. The Arabic text corpus was used, features are selected using FireFly and given to SVM for modeling. It was concluded that the proposed algorithm performed better than SVM.

In ref. [11], the authors had proposed Enhanced Cuckoo Search (ECS). It was then used for bloom filter optimization. This algorithm considers the weight of spam words. It outperformed the normal Cuckoo search.

2.2 Merits of Proposed Methodology:

- The paper has explored various machine learning techniques to resolve the problem of spam email classification.
- As the techniques were explored on various datasets, the results are considered to be more effective.
- The paper sheds light on optimization methods that are proven to be successful in producing promising results for classification of spam emails.
- Stochastic Gradient Descent was able to perform better among the base models as it is computationally fast and coverages faster and updates the parameters frequently for large datasets.
- This paper deals with datasets that are unbalanced, therefore stratified K- fold cross validation helps in balancing out the data into train and test set. It also validates the models' performance on multiple data folds.

2.3 Demerits of Proposed Methodology:

- The numerical dataset (PU) had limitations in feature extraction.
- The MLP classifier consumed more time and power.

2.4 Results

The performance measures of base models calculated for all the datasets are given in the following tables

	Precision	Recall	F1 Score	Accuracy
MNB	100	5.7	10.7	84.66
SGD	99.55	96.15	97.80	99.30
DT	87.01	88.24	87.48	95.89
RF	100	86.53	92.70	97.80
MLP	99.57	97.22	98.34	99.47

Table 2.4.1 Performance Measure- Ling Spam Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	100	8.61	15.62	85.04
SGD	99.33	91.37	95.16	98.49
DT	83.66	85.97	84.74	94.91
RF	97.27	86.75	91.61	97.44
MLP	99.36	93.98	96.58	98.91

Table 2.4.2 Performance Measure- Spam Assassin Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	98.86	98.91	98.94	98.92
SGD	98.16	99.73	98.94	98.91
DT	95.65	96.36	96.0	95.91
RF	98.37	99.18	98.77	98.75
MLP				

Table 2.4.3 Performance Measure- Enron Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	96.09	97.91	91.88	93.21
SGD	95.47	95.62	95.51	96.05
DT	90.60	87.08	88.59	90.18
RF	98.56	96.25	97.36	97.70
MLP	97.16	97.91	97.51	97.79

Table 2.4.4 Performance Measure- PU1 Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	nan	0.0	nan	80.28
SGD	91.75	87.14	88.72	95.77
DT	67.36	72.14	69.13	87.46
RF	97.08	67.14	78.34	93.09
MLP	96.98	90.00	92.96	97.32

Table 2.4.5 Performance Measure- PU2 Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	98.43	96.04	97.21	97.57
SGD	97.59	96.81	97.17	97.52
DT	89.68	90.10	89.86	91.04
RF	98.78	95.71	97.19	97.57
MLP	98.56	97.14	97.83	98.11

Table 2.4.6 Performance Measure- PU3 Dataset

	Precision	Recall	F1 Score	Accuracy
MNB	97.18	96.14	96.64	96.66
SGD	95.86	96.14	95.96	95.96
DT	86.77	86.31	86.52	86.57
RF	94.29	96.31	95.25	95.17
MLP	98.04	96.14	97.07	97.10

Table 2.4.7 Performance Measure- PUA Dataset

The accuracy of base models optimized with PSO and GA is given in the table 2.4.8

	PSO	GA
MNB	88.54	92.48
SGD	99.30	89.22
DT	96.18	96.17
RF	97.56	93.78
MLP	99.30	-

Table 2.4.8 Accuracy after optimizing the base models

CHAPTER 3

SOURCE CODE

3.1 SpamAssasin:

```
import numpy as np
import pandas as pd
import os
import zipfile as zf
import email
from bs4 import BeautifulSoup
import re
from tpot import TPOTClassifier
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

get_ipython().system('pip install tpot')

files = zf.ZipFile("spamassasin.zip", 'r')
files.extractall('spamass')
files.close()

def data_from_file():
    target=[]
    index=[]
    rows=[]
    flist = os.listdir("spamass\\spamassasin\\ham")
    for f in flist:
        ifile = open("spamass\\spamassasin\\ham\\"+f, encoding = "ISO-8859-1")
        rawtext=""

        rawtext = file_read(ifile)

        msg = email.message_from_string(rawtext)
        subject = str(msg['Subject'])

        body = email_parse_subject_body(rawtext)
```

```
subjectandbody=subject + "\n" + body

        rows.append({'text': subjectandbody, 'class': 0})
        index.append(f)
    flist = os.listdir("spamass\\spamassasin\\spam")

    for f in flist:
        ifile=open("spamass\\spamassasin\\spam\\" + f, encoding = "ISO-8859-1")
        rawtext=""
        rawtext = file_read(ifile)
        msg = email.message_from_string(rawtext)
        subject = str(msg['Subject'])
        body = email_parse_subject_body(rawtext)
        subjectandbody = subject + "\n" + body
        rows.append({'text': subjectandbody, 'class': 1})
        index.append(f)
    data_frame_from_email_and_class = pd.DataFrame(rows, index=index)
    return data_frame_from_email_and_class

def file_read(ifile):
    rawtext=""
    lines=ifile.readlines()
    for l in lines:
        rawtext = rawtext+l
    ifile.close()
    return rawtext

def preprocessing(htmltext):

    soup = BeautifulSoup(htmltext,"lxml")
    for script in soup(["script", "style"]):
        script.extract()
    email_text = soup.get_text()
    lines = (line.strip() for line in email_text.splitlines())
    group = (phrase.strip() for line in lines for phrase in line.split("  "))
    email_text = '\n'.join(group for group in group if group)
    sp_character = re.compile('(<|>|^|&|||_|-)')
    sp_character_removed = sp_character.sub('', email_text)
    return sp_character_removed

def email_parse_subject_body(rawtext):

    emailText = email.message_from_string(rawtext)
    maintype = emailText.get_content_maintype()
    if maintype == 'text':
        cleanedemail = emailText.get_payload()
    else:
        cleanedemail = ""
    preprocessed_email=preprocessing(cleanedemail)
```

```
return preprocessed_email

def model_performance(data_frame, targets, vectorizer, classifier):

    precision_list = []
    recall_list = []
    f1_list = []
    acc_list=[]


    #tpot = TPOTClassifier(generations=10, population_size=40, verbosity=2, random_state=42)

    #Folding data in K folds maintaining balanced spam and non-spam emails in training
    skf= StratifiedKFold(shuffle = True, n_splits = 10)
    x=data_frame['text']
    y=data_frame['class']
    skf.get_n_splits(x, y)
    for train_index, test_index in skf.split(x,y):

        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        x_train_features = vectorizer.fit_transform(x_train)
        x_test_features = vectorizer.transform(x_test)
        classifier.fit(x_train_features, y_train)
        y_predict = classifier.predict(x_test_features)

        confusion = confusion_matrix(y_test, y_predict)
        #print(confusion)
        #accuracy_on_training = accuracy_score(y_train, y_predict)
        #acc.append(accuracy_on_training)
        acc_value = accuracy(confusion)
        acc_list.append(acc_value)

        precision_value = precision_func(confusion)
        precision_list.append(precision_value)

        recall_value = recall_func(confusion)
        recall_list.append(recall_value)

        f1_value = f1_func(precision_value, recall_value)
        f1_list.append(f1_value)


    avgscore(precision_list, recall_list, f1_list, acc_list)

def avgscore(precision_list, recall_list, f1_list, acc_list):

    avg_precision = np.mean(precision_list, axis = 0)
    avg_recall = np.mean(recall_list, axis = 0)
    avg_f1 = np.mean(f1_list, axis = 0)
```

```
    avg_acc = np.mean(acc_list, axis=0)

    print("Average score of 10 folds \nPrecision:  ", avg_precision, "\nRecall:  ", avg_recall, "\nDF-
1: ", avg_f1, "\nAccuracy:", avg_acc, "\n",)

#precision calculation
def precision_func(confusion):
    precision = (confusion[1][1]) / (confusion[1][1] + confusion[0][1])
    return precision


#recall calculation
def recall_func(confusion):
    recall = confusion[1][1] / (confusion[1][1] + confusion[1][0])
    return recall
#DF-1 calculation
def f1_func(precision, recall):
    f1 = 2 * precision * recall/(precision + recall)
    return f1

#Accuracy calculation
def accuracy(conf):
    acc = (conf[0][0]+conf[1][1]) / (conf[0][0]+conf[0][1]+conf[1][0]+conf[1][1])
    return acc

data_frame = data_from_file()
targets = data_frame['class'].values
tfid_vectorizer = TfidfVectorizer()


naive_bayes=MultinomialNB()
print("\nNaive Bayes:\n")
model_performance(data_frame, targets, tfid_vectorizer, naive_bayes)

stochastic_gradient=SGDClassifier()
print("\nStochastic Gradient Descent:\n")
model_performance(data_frame, targets, tfid_vectorizer, stochastic_gradient)

dec_tree=DecisionTreeClassifier()
print("\nDecision Tree:\n")
model_performance(data_frame, targets, tfid_vectorizer, dec_tree)

rand_forest=RandomForestClassifier()
print("\nRandom Forest:\n")
model_performance(data_frame, targets, tfid_vectorizer, rand_forest)


mlp=MLPClassifier()
print("\nMultilayer Perceptron:\n")
model_performance(data_frame, targets, tfid_vectorizer, mlp)
```

3.2 PU:

```
import tarfile
import numpy as np
import pandas as pd
import os
import zipfile as zf
import email
from bs4 import BeautifulSoup
import re
from sklearn.metrics import accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
file = tarfile.open('PU123ACorpora.tar.gz')
file.extractall('PUA123')
file.close()
def file_read(ifile):
    rawtext=""
    lines=ifile.readlines()
    for l in lines:
        rawtext = rawtext+l
    ifile.close()
    return rawtext
def data_from_file():
    target = []
    index = []
    rows = []
    for i in range(1,11):
        flist = os.listdir("PUA123\\pu_corpora_public\\pu1\\part"+str(i))
        for text_file in flist:
            ifile = open("PUA123\\pu_corpora_public\\pu1\\part"+str(i)+"\\"+text_file)
            #print(text_file)
            raw_text = ""
            raw_text = file_read(ifile)
            if "spmsg" in text_file:
                rows.append({'text': raw_text, 'class':1})
            else:
                rows.append({'text': raw_text, 'class':0})          #legit
            index.append(text_file)
    data_frame = pd.DataFrame(rows, index=index)
    return data_frame
```

```
def model_performance(data_frame, targets, vectorizer, classifier):

    precision_list = []
    recall_list = []
    f1_list = []
    acc_list=[]

    #Folding data in K folds maintaining balanced spam and non-spam emails in training
    skf= StratifiedKFold(shuffle = True, n_splits = 10)
    x=data_frame['text']
    y=data_frame['class']
    skf.get_n_splits(x, y)

    acc1 = []

    for train_index, test_index in skf.split(x,y):

        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        x_train_features = vectorizer.fit_transform(x_train)
        x_test_features = vectorizer.transform(x_test)

        model = classifier.fit(x_train_features, y_train)
        prediction_on_training  = classifier.predict(x_test_features)
        confusion = confusion_matrix(y_test, prediction_on_training)
        acc_value = accuracy(confusion)
        acc_list.append(acc_value)

        precision_value = precision_func(confusion)
        precision_list.append(precision_value)

        recall_value = recall_func(confusion)
        recall_list.append(recall_value)

        f1_value = f1_func(precision_value, recall_value)
        f1_list.append(f1_value)

    avgscore(precision_list, recall_list, f1_list, acc_list)


#Average score for precision, recall and DF-1 calculation

def avgscore(precision_list, recall_list, f1_list, acc_list):

    avg_precision = np.mean(precision_list, axis = 0)
    avg_recall = np.mean(recall_list, axis = 0)
    avg_f1 = np.mean(f1_list, axis = 0)
    avg_acc = np.mean(acc_list, axis=0)
```

```
    print("Average score of 10 folds \nPrecision:  ", avg_precision, "\nRecall:  ", avg_recall, "\nF1
Score-1: ", avg_f1, "\nAccuracy:", avg_acc)



#precision calculation
def precision_func(confusion):

    precision = (confusion[1][1]) / (confusion[1][1] + confusion[0][1])
    return precision


#recall calculation
def recall_func(confusion):

    recall = confusion[1][1] / (confusion[1][1] + confusion[1][0])
    return recall


#DF-1 calculation
def f1_func(precision, recall):

    f1 = 2 * precision * recall/(precision + recall)
    return f1

def accuracy(conf):
    acc = (conf[0][0]+conf[1][1]) / (conf[0][0]+conf[0][1]+conf[1][0]+conf[1][1])
    return acc


data_frame = data_from_file()
targets = data_frame['class'].values
tfid_vectorizer = TfidfVectorizer()

naive_bayes=MultinomialNB()

stochastic_gradient=SGDClassifier()

dec_tree=DecisionTreeClassifier()

rand_forest=RandomForestClassifier()

mlp=MLPClassifier()

print("Naive Bayes:")
model_performance(data_frame, targets, tfid_vectorizer, naive_bayes)

print("Stochastic Gradient Descent:")
model_performance(data_frame, targets, tfid_vectorizer, stochastic_gradient)
```

```
print("Decision Tree:")
model_performance(data_frame, targets, tfid_vectorizer, dec_tree)

print("Random Forest:")
model_performance(data_frame, targets, tfid_vectorizer, rand_forest)

print("Multilayer Perceptron:")
model_performance(data_frame, targets, tfid_vectorizer, mlp)
```

3.3 PSO:

```
import numpy as np
import pandas as pd
import gradio as gr
import nltk
from nltk.corpus import stopwords
import pyswarms as ps
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from nltk.tokenize import sent_tokenize,word_tokenize
df = pd.read_csv('messages_1.csv')
df.head()
df.drop_duplicates(inplace=True)
df.drop(["subject"], axis=1, inplace=True)
x=df['message']
y=df['label']
model1 = MultinomialNB()
options = {'c1': 0.5, 'c2': 0.7, 'w':0.9}
no_of_particles = 10
acc1 = []
skf= StratifiedKFold(shuffle = True, n_splits = 10)
skf.get_n_splits(x, y)


def Obj_fun(x_train_features, x_test_features, y_train, y_test):
    model1.fit(x_train_features, y_train)
    predictone = model1.predict(x_train_features)
    acc1.append(model1.score(x_test_features, y_test))
    print(model1.score(x_test_features, y_test))
    #print("Of")
```

```
    return acc1


def pso():
    for train_index, test_index in skf.split(x, y):
        #x_train, x_test = data_frame['text'][train_index].values, data_frame['text'][test_index].values
        #y_train, y_test = targets[train_index], targets[test_index]
        x_train, x_test = x.iloc[train_index], x.iloc[test_index]
        y_train, y_test = y.iloc[train_index], y.iloc[test_index]
        x_train_features = feature_extraction.fit_transform(x_train)
        x_test_features = feature_extraction.transform(x_test)
        optimizer = ps.single.GlobalBestPSO(n_particles=no_of_particles,
dimensions=2,options=options)
        stats = optimizer.optimize(Obj_fun(x_train_features, x_test_features, y_train, y_test),iters=5)
        print(stats)

pso()

feature_extraction = TfidfVectorizer(min_df=1, stop_words='english',lowercase='True')
x_train_features = feature_extraction.fit_transform(x_train)
x_test_features = feature_extraction.transform(x_test)
#convert ytrain and ytest as integers
y_train=y_train.astype('int')
y_test=y_test.astype('int')

#acc = Obj_fun(x_train_features, x_test_features, y_train, y_test)
optimizer = ps.single.GlobalBestPSO(n_particles=no_of_particles,
dimensions=2,options=options)
stats = optimizer.optimize(Obj_fun(x_train_features, x_test_features, y_train, y_test),iters=5)
print(stats)
```

3.4 GA:

```
import numpy as np
import pandas as pd
#import gradio as gr
import nltk
from nltk.corpus import stopwords
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import confusion_matrix
from nltk.tokenize import sent_tokenize,word_tokenize

df = pd.read_csv('messages_1.csv')
df.drop_duplicates(inplace=True)
df.drop(["subject"], axis=1, inplace=True)
nltk.download('stopwords')

def process_data(text):
    no_punc = [char for char in text if (char not in string.punctuation)]
    no_punc = ''.join(no_punc)
    clean_words = [word for word in no_punc.split() if word.lower() not in
(stopword.words('english') and char.isalpha())]
    return clean_words

x=df['message']
y=df['label']

skf= StratifiedKFold(shuffle = True, n_splits = 10)
skf.get_n_splits(x, y)
feature_extraction = TfidfVectorizer(min_df=1, stop_words='english',lowercase='True')
for train_index, test_index in skf.split(x, y):
    #x_train, x_test = data_frame['text'][train_index].values, data_frame['text'][test_index].values
    #y_train, y_test = targets[train_index], targets[test_index]
    x_train, x_test = x.iloc[train_index], x.iloc[test_index]
    y_train, y_test = y.iloc[train_index], y.iloc[test_index]
    x_train_features = feature_extraction.fit_transform(x_train)
    x_test_features = feature_extraction.transform(x_test)

feature_extraction = TfidfVectorizer(min_df=1, stop_words='english',lowercase='True')
x_train_features = feature_extraction.fit_transform(x_train)
x_test_features = feature_extraction.transform(x_test)
#convert ytrain and ytest as integers
y_train=y_train.astype('int')
y_test=y_test.astype('int')

!pip install tpot
from tpot import TPOTClassifier

# Define the hyperparameter configuration space
parameters = {'alpha':range(1,100)}
# Set the hyperparameters of GA
ga1 = TPOTClassifier(generations= 10, population_size= 40, offspring_size= 20,
                     verbosity= 3, early_stop= 5,
                     config_dict=
                     {'sklearn.naive_bayes.MultinomialNB': parameters},
                     cv = 3, scoring = 'accuracy')
ga1.fit(x_train_features, y_train)
```

```
parameteres = {'loss':['hinge','log_loss', 'log'],
            'penalty':['l2', 'l1', 'elasticnet'],
            'alpha':0.0001,
            'max_iter':range(1,1000),
            'verbose':0,
            'epsilon':0.1}

ga2= TPOTClassifier(generations= 10, population_size= 40, offspring_size= 20,
                            verbosity= 3, early_stop= 5,
                            config_dict=
                            {'sklearn.linear_model.SGDClassifier': parameters},
                            cv = 3, scoring = 'accuracy')
ga2.fit(x_train_features, y_train)

parameters = {
       'n_estimators': range(20,200),
   "max_features":range(1,64),
   'max_depth': range(10,100),
   "min_samples_split":range(2,11),
   "min_samples_leaf":range(1,11),
   "criterion":['gini','entropy']
           }

ga3 = TPOTClassifier(generations= 10, population_size= 40, offspring_size= 20,
                            verbosity= 3, early_stop= 5,
                            config_dict=
                            {'sklearn.ensemble.RandomForestClassifier': parameters},
                            cv = 3, scoring = 'accuracy')
ga3.fit(x_train_features, y_train)


parameters = {'criterion':['gini', 'entropy', 'log_loss'],
            'splitter':['best', 'random']}

ga4 = TPOTClassifier(generations= 10, population_size= 40, offspring_size= 20,
                            verbosity= 3, early_stop= 5,
                            config_dict=
                            {'sklearn.tree.DecisionTreeClassifier': parameters},
                            cv = 3, scoring = 'accuracy')
ga4.fit(x_train_features, y_train)

parameters ={'activation':['identity', 'logistic', 'tanh', 'relu'],
           'solver':['bfgs', 'sgd', 'adam']}

ga5 = TPOTClassifier(generations= 10, population_size= 40, offspring_size= 20,
                            verbosity= 3, early_stop= 5,
                            config_dict=
                            {'sklearn.neural_network.MLPClassifier': parameters},
                            cv = 3, scoring = 'accuracy')
ga5.fit(x_train_features, y_train)
```

3.5 GUI:

```
import zipfile as zf
import numpy as np
import pandas as pd
import os
import email
import string
import nltk
nltk.download('stopwords')
from nltk.corpus import stopwords
from bs4 import BeautifulSoup
import re
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neural_network import MLPClassifier
from tkinter import *
from tkinter import filedialog
import tkinter.scrolledtext as st
from PIL import ImageTk, Image
from sklearn import *

def email_parse_subject_body(rawtext):

    emailText = email.message_from_string(rawtext)
    maintype = emailText.get_content_maintype()

    if maintype == 'text':
        cleanedemail = emailText.get_payload()
    else:
        cleanedemail = ""
    preprocessed_email=preprocessing(cleanedemail)

    return preprocessed_email
def file_read(ifile):
    rawtext=""
    lines=ifile.readlines()
    for l in lines:
        rawtext = rawtext+l
    ifile.close()
    return rawtext

def preprocessing(htmltext):

    soup = BeautifulSoup(htmltext,"lxml")
    for script in soup(["script", "style"]):
        script.extract()
```

```
    email_text = soup.get_text()
    lines = (line.strip() for line in email_text.splitlines())

    group = (phrase.strip() for line in lines for phrase in line.split("  "))
    email_text = '\n'.join(group for group in group if group)

    sp_character = re.compile('(<|>|^|&|||_|-)')
    sp_character_removed = sp_character.sub('', email_text)

    return sp_character_removed
df = pd.read_csv('messages_1.csv')
df.drop_duplicates(inplace=True)
df.drop(["subject"], axis=1, inplace=True)
def preprocess_text(message):
    """
    Takes in a string of text, then performs the following:
    1. Remove all punctuation
    2. Remove all stopwords
    3. Returns a list of the cleaned text
    """
    # Check characters to see if they are in punctuation
    without_punc = [char for char in message if char not in string.punctuation]

    # Join the characters again to form the string.
    without_punc = ''.join(without_punc)

    # Now just remove any stopwords
    return [word for word in without_punc.split() if word.lower() not in stopwords.words('english')]

df['message'].head().apply(preprocess_text)
from sklearn.feature_extraction.text import CountVectorizer
x_ls = df['message']
y_ls = df['label']
cv = CountVectorizer()
x_ls= cv.fit_transform(x_ls)


from sklearn.model_selection import train_test_split
x_ltrain, x_ltest, y_ltrain, y_ltest = train_test_split(x_ls, y_ls, test_size=0.20, random_state=0)


classifierl1 = MultinomialNB().fit(x_ltrain, y_ltrain)
classifierl2 = SGDClassifier().fit(x_ltrain, y_ltrain)
classifierl3 = DecisionTreeClassifier().fit(x_ltrain, y_ltrain)
classifierl4 = RandomForestClassifier().fit(x_ltrain, y_ltrain)
classifierl5 = MLPClassifier().fit(x_ltrain, y_ltrain)
file2 = zf.ZipFile("Enrons.zip", 'r')
file2.extractall('Enron')
file2.close()
```

```
def data_from_file_e():
    target=[]
    index=[]
    rows=[]
    for i in range(1,2):
        flist = os.listdir("Enron\\enron"+str(i)+"\\ham")
        for f in flist:
            ifile = open("Enron\\enron"+str(i)+"\\ham\\"+f,encoding="utf8", errors='ignore')
            rawtext=""

            rawtext = file_read(ifile)

            msg = email.message_from_string(rawtext)
            subject = str(msg['Subject'])

            body = email_parse_subject_body(rawtext)

            subjectandbody=subject + "\n" + body

            rows.append({'text': subjectandbody, 'class': 0})
            index.append(f)
        flist = os.listdir("Enron\\enron"+str(i)+"\\spam")

        for f in flist:
            ifile=open("Enron\\enron"+str(i)+"\\spam\\" + f, encoding="utf8", errors='ignore')
            rawtext=""
            rawtext = file_read(ifile)
            msg = email.message_from_string(rawtext)
            subject = str(msg['Subject'])
            body = email_parse_subject_body(rawtext)
            subjectandbody = subject + "\n" + body
            rows.append({'text': subjectandbody, 'class': 1})
            index.append(f)
    data_frame_from_email_and_class = pd.DataFrame(rows, index=index)
    return data_frame_from_email_and_class
data_frame_e = data_from_file_e()
from sklearn.feature_extraction.text import CountVectorizer
x_e=data_frame_e['text']
y_e=data_frame_e['class']
cv= CountVectorizer()
x_e = cv.fit_transform(x_e)

from sklearn.model_selection import train_test_split
x_etrain, x_etest, y_etrain, y_etest = train_test_split(x_e, y_e, test_size=0.20, random_state=0)

classifiere1 = MultinomialNB().fit(x_etrain, y_etrain)
classifiere2 = SGDClassifier().fit(x_etrain, y_etrain)
classifiere3 = DecisionTreeClassifier().fit(x_etrain, y_etrain)
classifiere4 = RandomForestClassifier().fit(x_etrain, y_etrain)
classifiere5 = MLPClassifier().fit(x_etrain, y_etrain)
```

```
root = Tk()
root.title('SPAM DETECTOR')
root.geometry('1500x1500')
root.config(bg='white')

bgimg = Image.open("try3.jpg")
bgimg = bgimg.resize((1500,750))
bgimg=ImageTk.PhotoImage(bgimg)
bglabel = Label(root, image=bgimg)
bglabel.image = bgimg
bglabel.place(x=0, y=0)

heading = Label(root, text='Spam Email Detection with Bio-optimized Machine Learning Algorithms',
            font=('comic sans ms', 25, 'bold'), height=2)
heading.grid(row=0, column=0, columnspan=5)

img = Image.open("logo.png")
img = img.resize((150, 92))
img = ImageTk.PhotoImage(img)
imglabel = Label(root, image=img)
imglabel.image = img
imglabel.grid(row=0, column=5)

datalabel = Label(root, text='Select Dataset', font=('comic sans ms', 15, 'bold'))
datalabel.grid(row=1, column=0)


def datadisplay(choice):
   choice = variable1.get()
   datadesc.delete('1.0', END)
   if choice == "Enron":
      datadesc.insert(INSERT, "Enron dataset includes 6 separate datasets that contain 3000-4000 individual emails as text files.The dataset includes numbers, alphabets and characters.The total number of emails in this dataset is 36715 (Spam-20170, Ham-16545).This dataset was published in the year 2006.")
   elif choice == "Lingspam":
      datadesc.insert(INSERT, "Ling-Spam dataset is divided into 10 parts from the ‘bare’ distribution that includes individual emails as a text file (.txt). This data is not pre-processed, and it includes numbers, alphabets and characters The total number of emails in this dataset is 2893(Spam-481, Ham-2412).This dataset was published in the year 2000.")
   # print(choice+" is selected")

def modeldisplay(choice):
   datasetchoice = variable1.get()
   modelchoice = variable2.get()
   modeldesc.delete('1.0', END)
   if datasetchoice == "Enron" :
      if modelchoice == "MNB":
         modeldesc.insert(INSERT, "Training accuracy of MNB = 99.24 ")
```

```
elif modelchoice == "SGD":
            modeldesc.insert(INSERT, "Training accuracy of  SGD= 99.44 ")
        elif modelchoice == "DT":
            modeldesc.insert(INSERT, "Training accuracy of  DT= 100 ")
        elif modelchoice == "RF":
            modeldesc.insert(INSERT, "Training accuracy  RF= 100 ")
        else:
            modeldesc.insert(INSERT, "Training accuracy of  MLP= 100")

    elif datasetchoice == "Lingspam":
        if modelchoice == "MNB":
            modeldesc.insert(INSERT, "Training accuracy of MNB = 92.62")
        elif modelchoice == "SGD":
            modeldesc.insert(INSERT, "Training accuracy of SGD = 100")
        elif modelchoice == "DT":
            modeldesc.insert(INSERT, "Training accuracy of DT= 94.39")
        elif modelchoice == "RF":
            modeldesc.insert(INSERT, "Training accuracy of RF = 100")
        else:
            modeldesc.insert(INSERT, "Training accuracy of MLP= 100")


def upload_mail():
    mailtext.delete('1.0', END)
    file = filedialog.askopenfilename()
    fob = open(file, 'r')
    #print(fob.read())
    mailtext.insert(END, fob.read())

def predict():
    #lab = ['not spam', 'spam']
    datasetchoice = variable1.get()
    modelchoice = variable2.get()
    xp = cv.transform([mailtext.get("1.0", END)]).toarray()
    #xp.reshape(-1,1)
    #print(x.shape)
    if datasetchoice == "Enron":
        if modelchoice == "MNB":
            p = classifiere1.predict(xp)
        elif modelchoice == "SGD":
            p = classifiere2.predict(xp)
        elif modelchoice == "DT":
            p = classifiere3.predict(xp)
        elif modelchoice == "RF":
            p = classifiere4.predict(xp)
        elif modelchoice == "MLP":
            p = classifiere5.predict(xp)
        s = [str(i) for i in p]
        a = int("".join(s))
```

```
        if a == 0:
            hamimg = Image.open("ham.png")
            hamimg = hamimg.resize((100, 75))
            hamimg = ImageTk.PhotoImage(hamimg)
            hamimglabel = Label(root, image=hamimg)
            hamimglabel.image = hamimg
            hamimglabel.grid(row=3, column=2)
        elif a == 1:
            spamimg = Image.open("spam.png")
            spamimg = spamimg.resize((100, 75))
            spamimg = ImageTk.PhotoImage(spamimg)
            spamimglabel = Label(root, image=spamimg)
            spamimglabel.image = spamimg
            spamimglabel.grid(row=3, column=2)

    elif datasetchoice == "Lingspam":
        if modelchoice == "MNB":
            p = classifierl1.predict(xp)
        elif modelchoice == "SGD":
            p = classifierl2.predict(xp)
        elif modelchoice == "DT":
            p = classifierl3.predict(xp)
        elif modelchoice == "RF":
            p = classifierl4.predict(xp)
        elif modelchoice == "MLP":
            p = classifierl5.predict(xp)
        s = [str(i) for i in p]
        a = int("".join(s))
        if a == 0:
            hamimg = Image.open("ham.png")
            hamimg = hamimg.resize((150, 100))
            hamimg = ImageTk.PhotoImage(hamimg)
            hamimglabel = Label(root, image=hamimg)
            hamimglabel.image = hamimg
            hamimglabel.grid(row=3, column=2)
        elif a == 1:
            spamimg = Image.open("spam.png")
            spamimg = spamimg.resize((150, 100))
            spamimg = ImageTk.PhotoImage(spamimg, image=spamimg)
            spamimglabel = Label(root)
            spamimglabel.image = spamimg
            spamimglabel.grid(row=3, column=2)



def compare():
    datasetchoice = variable1.get()
    modelchoice = variable2.get()

    fivefoldtxt.delete('1.0', END)
```

```
    fivefoldtxt.insert(INSERT,"\n5-FOLD CROSS VALIDATION\n")
    tenfoldtxt.delete('1.0', END)
    tenfoldtxt.insert(INSERT,"\n10-FOLD CROSS VALIDATION\n")

    if datasetchoice == "Enron" :
        if modelchoice == "MNB":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :87.78 \nPrecision :100 \nRecall :57.86 \nF1-
Score :73.28 ")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :98.92 \nPrecision :98.96 \nRecall :98.91 \nF1-
Score :98.94")
        elif modelchoice == "SGD":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :98.78 \nPrecision :97.05 \nRecall :98.80 \nF1-
Score:97.91")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :98.91 \nPrecision :98.16 \nRecall :99.73 \nF1-
Score :98.94")
        elif modelchoice == "DT":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :94.39 \nPrecision :89.79 \nRecall :90.99 \nF1-
Score:90.38")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :95.91 \nPrecision :95.65 \nRecall :96.36 \nF1-
Score :96.00")
        elif modelchoice == "RF":
            fivefoldtxt.insert(INSERT, "\nTest accuracy : - \nPrecision : -  \nRecall : -\nF1-Score : -")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :98.75 \nPrecision :98.37 \nRecall :99.18 \nF1-
Score :98.77")
        else:
            fivefoldtxt.insert(INSERT, "\nTest accuracy : -  \nPrecision : - \nRecall : - \nF1-Score :- ")
            tenfoldtxt.insert(INSERT, "\nTest accuracy : - \nPrecision : -  \nRecall : - \nF1-Score : -")

    elif datasetchoice == "Lingspam":
        if modelchoice == "MNB":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :84.52 \nPrecision :100 \nRecall :4.9 \nF1-
Score :9.2 ")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :84.66 \nPrecision :100 \nRecall :5.7 \nF1-
Score :10.7 ")
        elif modelchoice == "SGD":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :99.37 \nPrecision :99.78 \nRecall :96.36 \nF1-
Score:98.03")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :99.30 \nPrecision :99.55 \nRecall :96.15 \nF1-
Score :97.80")
        elif modelchoice == "DT":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :95.82 \nPrecision :88.02 \nRecall :86.34 \nF1-
Score:87.06")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :95.89 \nPrecision :87.01 \nRecall :88.21 \nF1-
Score :87.48")
        elif modelchoice == "RF":
            fivefoldtxt.insert(INSERT, "\nTest accuracy :97.77 \nPrecision :99.75 \nRecall :86.53 \nF1-
Score:92.61")
            tenfoldtxt.insert(INSERT, "\nTest accuracy :97.80 \nPrecision :100 \nRecall :86.53 \nF1-
Score :92.70 ")
```

```
else:
        fivefoldtxt.insert(INSERT, "\nTest accuracy :99.58 \nPrecision :99.56 \nRecall :97.86 \nF1-Score:98.70")
        tenfoldtxt.insert(INSERT, "\nTest accuracy 99.47: \nPrecision :99.57 \nRecall :97.22 \nF1-Score :98.34")


dataset = ['Lingspam', 'Enron']
model = ['MNB', 'SGD', 'DT', 'RF', 'MLP']
variable1 = StringVar()
variable1.set(dataset[0])
variable2 = StringVar()
variable2.set(model[0])

datadropdown = OptionMenu(root, variable1, *dataset, command=datadisplay)
datadropdown.grid(row=1, column=1)

datadesc = Text(root, height=8, width=50)
datadesc.grid(row=1, column=2)

modellabel = Label(root, text='Select Model', font=('comic sans ms', 15, 'bold'))
modellabel.grid(row=2, column=0)

modeldropdown = OptionMenu(root, variable2, *model, command=modeldisplay)
modeldropdown.grid(row=2, column=1)

modeldesc = Text(root, height=5, width=50)
modeldesc.grid(row=2, column=2)

mailbutton = Button(root, text='Upload Mail', font=('comic sans ms', 15, 'bold'), command=upload_mail)
mailbutton.grid(row=2, column=3)

mailtext = st.ScrolledText(root, undo=True, font=('comic sans ms', 10), height=10, width=30)
mailtext.grid(row=2, column=4, columnspan=2, rowspan=2)

predictbutton = Button(root, text='Predict', font=('comic sans ms', 15, 'bold'), command=predict)
predictbutton.grid(row=3, column=3)

comparebutton =Button(root, text='Compare', font=('comic sans ms', 15, 'bold'), command=compare)
comparebutton.grid(row=4, column=0)

fivefoldtxt = Text(root, height=10, width=50)
fivefoldtxt.grid(row=4, column=1, columnspan=2)

tenfoldtxt = Text(root, height=10, width=50)
tenfoldtxt.grid(row=4, column=3, columnspan=2)
root.mainloop()
```

CHAPTER 4

SNAPSHOTS

The GUI is developed in Tkinter. Fig. 4.1 shows the GUI predicting spam email and Fig. 4.2 depicts the GUI predicting ham email

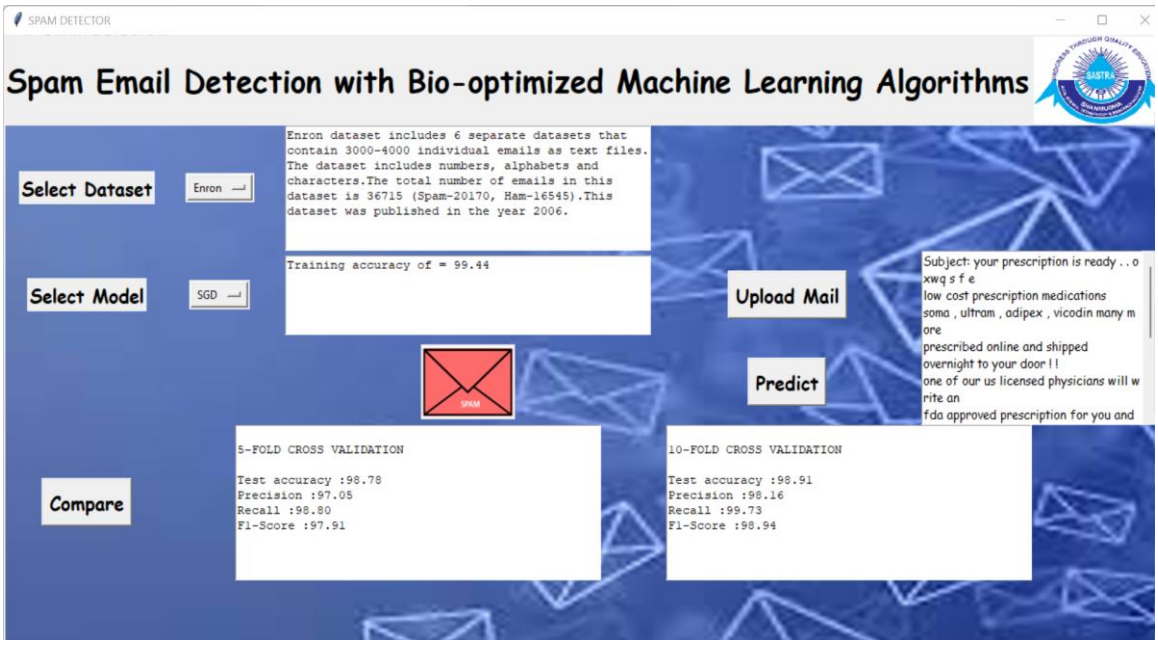


Fig 4.1 Spam Detector GUI predicting spam email

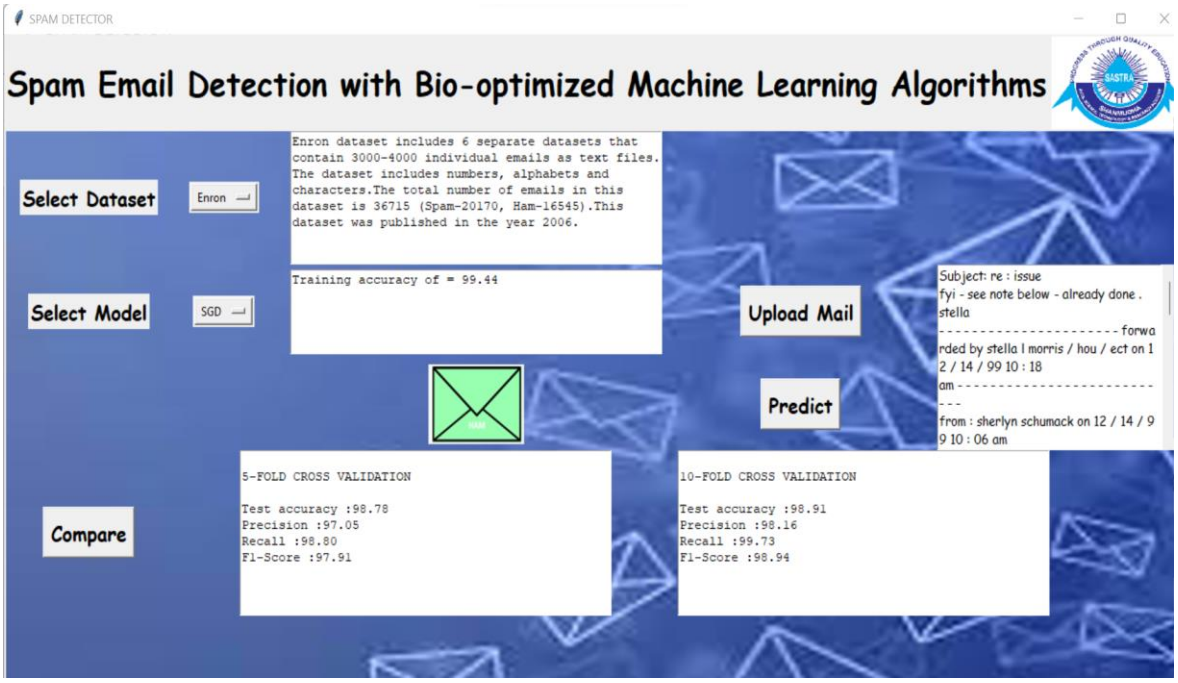


Fig 4.2 Spam Detector GUI predicting ham email

CHAPTER 5

CONCLUSION AND FUTURE PLANS

Models integrated with bio-inspired algorithms were effectively incorporated in the project. The experiment employed a spam email corpus that was both alphabetical as well as numerical. The numerical datasets had limitations with feature extraction as the words are vectorized into numbers. The alphabetical datasets showed better performance with feature extraction and outcome prediction. The five base machine learning algorithms were experimented with Scikit - learns library modules. And the results were further optimized with PSO and GA. It was inferred that GA performed better both numerical and alphabetical corpuses. MNB and SGD performed better when optimized with PSO, Random Forest and Decision Tree performed well when optimized with GA. In terms of performance measures (i.e.), F1- Score, Precision, Recall and Accuracy, GA had more impact than PSO. It was concluded that MNB is the most appropriate machine learning algorithm for spam classification.

5.1 Future Plans

1. Implementing Firefly, Bee Colony and Ant Colony Optimization for optimization with the base models.
2. Implementing MLP with GPU application using Keras and TensorFlow.
3. Improving accuracy by integrating techniques of NLP.

CHAPTER 6

REFERENCES

[1]. W. Feng, J. Sun, L. Zhang, C. Cao, and Q. Yang, ‘‘A support vector machine based Naive Bayes algorithm for spam filtering,’’ in Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC), Dec. 2016, pp. 1–8, doi: 10.1109/pccc.2016.7820655.

[2]. W. Awad and S. ELseuofi, ‘‘Machine learning methods for spam E-Mail classification,’’ Int. J. Comput. Sci. Inf. Technol., vol. 3, no. 1, pp. 173–184, Feb. 2011, doi: 10.5121/ijcsit.2011.3112.

[3]. S. Mohammed, O. Mohammed, and J. Fiaidhi, ‘‘Classifying unsolicited bulk email (UBE) using Python machine learning techniques,’’ Int. J. Hybrid Inf. Technol., vol. 6, no. 1, pp. 43–55, 2013. [Online].

[4]. A. Wijaya and A. Bisri, ‘‘Hybrid decision tree and logistic regression classifier for email spam detection,’’ in Proc. 8th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE), Oct. 2016, pp. 1–4, doi: 10.1109/ICITEED.2016. 7863267.

[5]. E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E.Ajibuwa, ‘‘Machine learning for email spam filtering: Review, approaches and open research problems,’’ Heliyon, vol. 5, no. 6, Jun. 2019, Art. no. e01802, doi: 10.1016/j.heliyon.2019.e01802.

[6]. K. Agarwal and T. Kumar, ‘‘Email spam detection using integrated approach of Naïve Bayes and particle swarm optimization,’’ in Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS), Jun. 2018, pp. 685–690, doi: 10.1109/ICCONS.2018.8662957.

[7]. R. Belkebir and A. Guessoum, ‘‘A hybrid BSO-Chi2-SVM approach to arabic text categorization,’’ in Proc. ACS Int. Conf. Comput. Syst. Appl. (AICCSA), Ifran, Morocco, May 2013, pp. 1–7, doi: 10.1109/ AICCSA.2013.6616437.

[8]. A. I. Taloba and S. S. I. Ismail, ‘‘An intelligent hybrid technique of decision tree and genetic algorithm for E-Mail spam detection,’’ in Proc. 9th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS), Cairo, Egypt, Dec. 2019, pp. 99–104, doi: 10.1109/ICICIS46948.2019.9014756.

[9]. R. Karthika and P. Visalakshi, ‘‘A hybrid ACO based feature selection method for email spam classification,’’ *WSEAS Trans. Comput, vol. 14, pp. 171–177,* 2015. [Online].

[10]. S. L. Marie-Sainte and N. Alalyani, ‘‘Firefly algorithm based feature selection for Arabic text classification,’’ *J. King Saud Univ.-Comput. Inf. Sci., vol. 32, no. 3, pp. 320–328,* Mar. 2020. [Online].

[11]. E. A. Natarajan, S. Subramanian, and K. Premalatha, ‘‘An enhanced cuckoo search for optimization of bloom filters in spam filtering,’’ *Global J. Comput. Sci. Technol., vol. 12, no. 1, pp. 75–81,* 2012. Accessed: Jan. 18, 2020. [Online].

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 24, 2020, accepted October 2, 2020, date of publication October 13, 2020, date of current version October 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3030751

Detecting Spam Email With Machine Learning Optimized With Bio-Inspired Metaheuristic Algorithms

SIMRAN GIBSON[1], BIJU ISSAC[1], (Senior Member, IEEE), LI ZHANG[1], (Senior Member, IEEE), AND SEIBU MARY JACOB[2], (Member, IEEE)

[1]Department of Computer and Information Sciences, Northumbria University, Newcastle upon Tyne NE1 8ST, U.K.
[2]School of Computing, Engineering, and Digital Technologies, Teesside University, Middlesbrough TS1 3BX, U.K.

Corresponding author: Biju Issac (bissac@ieee.org)

ABSTRACT Electronic mail has eased communication methods for many organisations as well as individuals. This method is exploited for fraudulent gain by spammers through sending unsolicited emails. This article aims to present a method for detection of spam emails with machine learning algorithms that are optimized with bio-inspired methods. A literature review is carried to explore the efficient methods applied on different datasets to achieve good results. An extensive research was done to implement machine learning models using Naïve Bayes, Support Vector Machine, Random Forest, Decision Tree and Multi-Layer Perceptron on seven different email datasets, along with feature extraction and pre-processing. The bio-inspired algorithms like Particle Swarm Optimization and Genetic Algorithm were implemented to optimize the performance of classifiers. Multinomial Naïve Bayes with Genetic Algorithm performed the best overall. The comparison of our results with other machine learning and bio-inspired models to show the best suitable model is also discussed.

INDEX TERMS Machine learning, bio-inspired algorithms, cross-validation, particle swarm optimization, genetic algorithm.

I. INTRODUCTION

Machine learning models have been utilized for multiple purposes in the field of computer science from resolving a network traffic issue to detecting a malware. Emails are used regularly by many people for communication and for socialising. Security breaches that compromises customer data allows 'spammers' to spoof a compromised email address to send illegitimate (spam) emails. This is also exploited to gain unauthorized access to their device by tricking the user into clicking the spam link within the spam email, that constitutes a phishing attack [1].

Many tools and techniques are offered by companies in order to detect spam emails in a network. Organisations have set up filtering mechanisms to detect unsolicited emails by setting up rules and configuring the firewall settings. Google is one of the top companies that offers 99.9% success in detecting such emails [2]. There are different areas for deploying the spam filters such as on the gateway (router), on the cloud hosted applications or on the user's computer. In order to overcome the detection problem of spam emails, methods such as content-based filtering, rule-based filtering or Bayesian filtering have been applied.

Unlike the 'knowledge engineering' where spam detection rules are set up and are in constant need of manual updating thus consuming time and resources, Machine learning makes it easier because it learns to recognise the unsolicited emails (spam) and legitimate emails (ham) automatically and then applies those learned instructions to unknown incoming emails [2].

The proposed spam detection to resolve the issue of the spam classification problem can be further experimented by feature selection or automated parameter selection for the models. This research conducts experiments involving five different machine learning models with Particle Swarm Optimization (PSO) and Genetic Algorithm (GA). This will be compared with the base models to conclude whether the proposed models have improved the performance with parameter tuning.

The associate editor coordinating the review of this manuscript and approving it for publication was Hisao Ishibuchi.

The rest of this article is organised as follows: Section II presents the research to identify techniques and methods used to resolve the classification problem. This is followed by section III that introduces the proposed work. Section IV explains the tools and implementation techniques. Section V introduces the Machine Learning algorithms that are implemented followed by section VI that explains the structure of the Python program, datasets and requirements. Section VIII discusses on the results of base model on datasets. Section IX explains the tuning of parameters. Section X explains the PSO and GA integration. The results of the optimized classifiers on different datasets are described in section XI, followed by comparison and evaluation in section XII. Section XIII and XIV talks about the future implementation and conclusion.

II. RELATED WORK

A. MACHINE LEARNING

Researchers have taken a lead to implement machine learning models to detect spam emails. In the paper [3], the authors have conducted experiments with six different machine learning algorithms: Naïve Bayes (NB) classification, K-Nearest Neighbour (K-NN), Artificial Neural Network (ANN), Support Vector Machine (SVM), Artificial Immune System and Rough Sets. Their aim of the experiment was to imitate the detecting and recognising ability of humans. Tokenisation was explored and the concept provided two stages: Training and Filtering. Their algorithm consisted of four steps: Email Pre-Processing, Description of the feature, Spam Classification and Performance Evaluation. It concluded that the Naïve Bayes provided the highest accuracy, precision and recall.

Feng *et al.* [1] describes a hybrid system between two machine learning algorithms i.e. SVM-NB. Their proposed method is to apply the SVM algorithm and generate the hyperplane between the given dimensions and reduce the training set by eliminating datapoints. This set will then be implemented with NB algorithm to predict the probability of the outcome. This experiment was conducted on Chinese text corpus. They successfully implemented their proposed algorithm and there was an increase in accuracy when compared to NB and SVM on their own.

Mohammed *et al.* [4] aimed to detect the unsolicited emails by experimenting with different classifiers such as: NB, SVM, KNN, Tree and Rule based algorithms. They generated a vocabulary of Spam and Ham emails which is then used to filter through the training and testing data. Their experiment was conducted with Python programming language on Email-1431 dataset. They concluded that NB was the best working classifier followed by Support Vector Machine.

Wijaya and Bisri [5] proposes a hybrid-based algorithm, which is integrating Decision Tree with Logistic Regression along with False Negative threshold. They were successful in increasing the performance of DT. The results were compared with the prior research. The experiment was conducted on the SpamBase dataset. The proposed method presented a 91.67% accuracy.

B. BIO-INSPIRED METHODS

Agarwal and Kumar [6] experimented with NB along with Particle Swarm Optimisation (PSO) technique. The paper used the emails from Ling-Spam corpus and aimed to acquire an improvement in F1-score, Precision, Recall and Accuracy. The paper used Correlation Feature Selection (CFS) to select appropriate features from the dataset. The dataset was split into 60:40 ratio. Particle Swarm Optimisation was integrated along with Naïve Bayes. They concluded a success when their proposed integrated method increased the accuracy of the detection compared to NB alone [6].

Belkebir and Guessoum [7] reviewed the SVM algorithm along with Bee Swarm Optimization (BSO) and Chi-Squared on Arabic Text. Since there have been plenty of research conducted for text mining on English and some European languages, the authors considered to review the algorithms work on Arabic language. They experimented with three different approaches to categorise automatic text – Neural networks, Support Vector Machine (SVM) and SVM optimizing with Bee Swarm Algorithm (BSO) along with Chi-Squared. Bee Swarming Optimization algorithm is inspired by the behaviour of swarm of bees to achieve global solution. A search area is divided and each area within the divided section is assigned to other bees to explore. Every solution is distributed amongst the bees and the best solution is accepted and the process is repeated until the solution meets the criteria of the problem.

The main problem advertised is: ''The problem of selecting the set of attributes is NP-hard''. The research explains the problem dealing with the feature selection due to the computation time. A vocabulary is generated and fed into the Chi2-BSO algorithm to acquire the features and finally the achieved result is loaded within the SVM algorithm. The experiment was carried on OSAC dataset which included 22,429 text records. The study randomly selected 100 texts from each category distributed by 70:30 ratio. The program performed removal of digits, Latin alphabets, isolated letters, punctuation marks and stopwords. The document representation step was conducted with different modes for all approaches – SVM, BSO-CHI-SVM and artificial neural network (ANN). The SVM outperformed the ANN execution time. The proposed algorithm BSO-CHI-SVM exceeds the learning time but it is still identified as effective [7]. The paper concluded that the proposed algorithm provides an accuracy rate of 95.67%. They have also stated that SVM approach outperformed ANN. A further development is to evaluate the approach of this article on other datasets and use modes such as n-gram or concept representation.

Many researchers have also researched the human evolutionary processes to optimize the ML algorithm's performance. Taloba and Ismail [8] explored Genetic Algorithm (GA) optimization by integrating it with Decision Tree (DT). The authors recognise the overfitting problem with dimension of feature space and attempt to overcome this issue by feature extraction with Principle Component Analysis (PCA). The paper provides an intensive background

of algorithms used and proceeds with proposed algorithm. Their program performs pre-processing, feature weighing and feature extraction. The proposed algorithm is to find the optimal value of the parameter provided for the Decision tree (DT) algorithm. The DT algorithm used is J-48 to generate the rules and then apply GA with fitness function to obtain the accuracy. The program uses the BLX-α for fitness search and performance. Their fitness function was conducted on each individual of GA. The experiment was conducted with the Enron spam dataset. The paper concluded that the GADT proposed algorithm provided higher accuracy when compared with other classifiers without PCA. Another experiment compared the performance measurement with using the PCA which provided higher accuracy than GADT itself.

Karthika and Visalakshi [9] reviews the ML algorithm – SVM along with the optimization technique – Ant Colony Optimization (ACO). The proposed algorithm was performed on the SpamBase dataset with supervised learning method. The paper briefly defines the existing work based on pheromone updating and fitness function. The paper provides an overview of the ML algorithm such as NB, SVM and KNN classifiers. The proposed algorithm was conducted by integrating the ACO algorithm into the SVM ML algorithm. ACO is based on the behaviour of the ants observed while creating a shortest path towards the food source. The paper states that the proposed ACO based feature selection algorithm deducts the memory requirement along with the computational time. The experiment uses the N-fold cross validation technique to evaluate the datasets with different measures. The feature selection methods were used with the ACO. The result of the proposed algorithm ACO-SVM was higher than the rest of the ML algorithms itself. The paper concluded that the accuracy of ACO-SVM was 4% higher than the SVM itself alone. The paper evaluated that the optimization algorithm resolves the activities of the problem simultaneously to classify the emails into ham and spam [9].

Additional research looked at algorithms for optimization such as Firefly and Cuckoo search. The Firefly algorithm in the paper [10] was used with SVM. The researchers experimented with the Arabic text with feature selection. The paper concluded that the proposed method outperforms the SVM itself. The paper [11], proposes Enhanced Cuckoo Search (ECS) for bloom filter optimization. This is where the weight of the spam word is considered. It was concluded that their proposed optimization technique of ECS outperforms the normal Cuckoo search.

The work in the above research has provided an insight into hybrid systems as well as optimization techniques. The bio-inspired techniques show more promising results in terms of accurately detecting a spam email.

III. PROPOSED WORK

This research will experiment Bio-inspired algorithms along with Machine learning models. This will be conducted on different spam email corpora that are publicly available. The paper aims to achieve the following objectives:

1) To explore machine learning algorithms for the spam detection problem.
2) To investigate the workings of the algorithms with the acquired datasets.
3) To implement the bio-inspired algorithms.
4) To test and compare the accuracy of base models with bio-inspired implementation.
5) To implement the framework using Python.

Scikit-Learn library will be explored to perform the experiments with Python, and this will enable to edit the models, conduct pre-processing and calculate the results. The program scripts will be implemented further with the optimization techniques and compared with the base results i.e with default parameters.

The spam detection engine should be able to take email datasets as input and with the help of text mining and optimized supervised algorithms, it should be able to classify the the email as ham or spam. Figure-1 represents the process that is followed to implement the model.

IV. TOOLS AND TECHNIQUES

Some of the tools and techniques used in this work are discussed below.

A. WEKA

WEKA is a GUI tool that allows to load a dataset and apply different functions/rules upon an algorithm [51]. The application allows to apply the classification, regression, clustering algorithms and enable to visualise the data and the performance of the algorithm. An '.arff' file format of the spam datasets were fed into the program.

Table-1 provides the average accuracy taken from the datasets for each algorithm within WEKA. The highest accuracy was provided by Multinomial Naïve Bayes (MNB), SMO, J48 and Random Forest. Three Naïve Bayes algorithms were tested using WEKA and MNB was the better amongst the three.

TABLE 1. WEKA results.

Classifiers	Average
IBK	85.79%
OneR	81.91%
Naïve Bayes	90.46%
Naïve Bayes Multinomial	92.65%
SMO	93.98%
AdaBoost	89.48%
Bagging	89.37%
ZeroR	63.07%
Decision Stump	81.33%
Hoeffding Tree	84.33%
J48	89.53%
Random Forest	93.04%
Random Tree	83.13%
Naïve Bayes Multinomial Text	63.07%

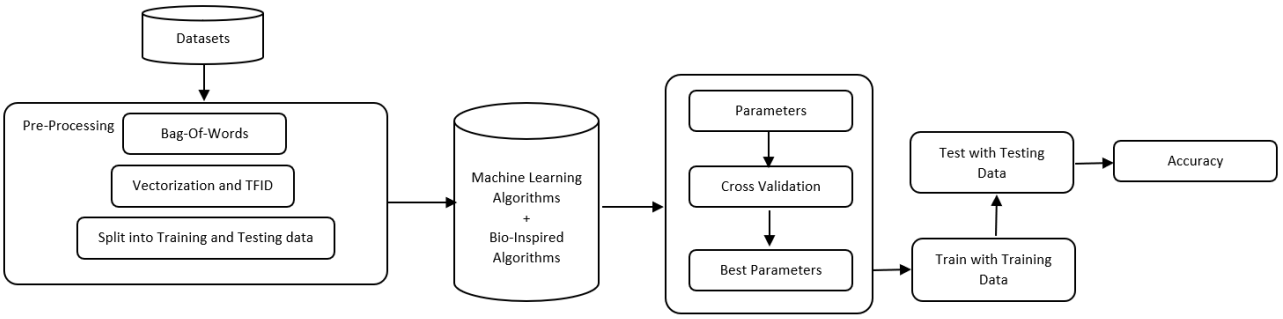


FIGURE 1. Spam detection block diagram.

In this experiment WEKA acted as a black box and provided the better performing algorithms which were Support Vector, Random Forest, Naïve Bayes and Decision Tree.

Since spam email detection falls into classification category, supervised learning method will be used. Supervised learning is a concept where the dataset is split into two parts: 1) Training data and 2) Testing data. The main aim of this learning method is to train a classifier with a given data and parameters and then predict the outcome with the testing dataset which will not be known to the program or classifier [12].

The models will be trained with a training dataset of 60%, 70%, 75% and 80%. Once the model is trained, model will be provided with the testing dataset which is distributed as 40%, 30%, 25% and 20% respectively with training dataset. This will provide a better knowledge of what percentage split is best suited and thus be more efficient to work with majority of the datasets. This will provide results on classifiers working best with more or less training data.

B. SCIKIT-LEARN

Scikit-Learn (SKLearn) is an environment that is incorporated with Python programming language. The library offers a wide range of supervised algorithms that will be suitable for this project [13]. The library offers high-level implementation to train with the 'Fit' methods and 'predict' from an estimator (Classifier). It also offers to perform the cross validation, feature selection, feature extraction and parameter tuning [14].

C. KERAS

Keras is an API that supports Neural Networks. The API supports other deep learning algorithms for easy and fast approach. It offers CPU and GPU running capabilities in order to simultaneously process the models. Online tutorials are available for neural network for learning and development. Their guide demonstrates the performance optimization techniques to utilize GPU and ways to work with RNN algorithm and other deep learning algorithms [15].

D. TensorFlow

Tensorflow is an end-to-end ML platform that is developed by Google. The architecture lets a user run the program on multiple CPUs and it also has access to GPUs. The website also provides a learning platform for both beginners and experts. TensorFlow can also be incorporated with Keras to perform deep learning experiments [16].

E. PYTHON PLATFORMS

Research was conducted into the different platforms that could be used for ML program implementation in Python.

1) SPYDER

Spyder is an Integrated Development Environment platform for Python programming language [17]. Spyder is incorporated within the Anaconda framework. The software allows the user to investigate the workings of a program. The program is capable to include multiple panels such as 'console' where the output can be seen, 'Variable Explorer' where the assignment of the variables can be investigated, 'Editor' to edit the program and other panels such as 'File Explorer' and 'History'.

2) JUPYTER NOTEBOOK

This is an open source tool that provides a Python framework. This is similar to 'Spyder' IDE, except this tool lets a user run the source code via a web browser [18]. Anaconda framework also offers 'Jupyter' to be utilised by the user through the local server.

F. ONLINE PLATFORMS

Along with the desktop-based platforms, other online platforms that offers additional support are: Google Collaboratory and Kaggle. Both platforms are the top ML and DL based that also offers TPU (Tensor Processing Unit) [19] along with CPU and GPU. Multiple core servers can also be accessed. The platforms are cloud-based, and the user's program is run until the 'Runtime' is ended.

V. MACHINE LEARNING MODELS

The subsections below explain each of the Machine Learning models that will be implemented to achieve the aim of this work. The sections are accompanied with mathematical equations along with the pseudocode algorithms. The algorithms

define the variables ''TrX'' as a Training subset of ''X'' and ''TeX'' as Testing subset.

A. Naïve BAYES - MULTINOMIAL

Naïve Bayes model is used to resolve classification problems by using probability techniques. The Naïve Bayes algorithm for this article can be denoted as equation-1 [20]:

$$\text{P(Class|WORD)} = \frac{\text{(P(WORD|Class)} \times \text{P(Class))}}{\text{(P(WORD))}} \quad (1)$$

where WORD is ($word_1, word_2, \ldots word_n$) from within an uploaded email and 'Class' is either 'Spam' or 'Ham'. The algorithm calculates the probability of a class from the bag of words provided by the program. Where P(Class | WORD) is a posterior probability, P(WORD | Class) is likelihood and P(Class) is the prior probability [21].

If 'Class' = Spam, the equation could be rewritten to find the spam email from the given words, and this can be further simplified as equation-2:

$$\text{P(Class|WORD)} = \frac{\Gamma_{\text{i=1}}^{\text{n}}\text{P(word\_i|Spam)} \times \text{P(Spam)}}{\text{P(word\_1, word\_2, \ldots word\_n)}} \quad (2)$$

There are three types of Naïve Bayes algorithms: Multinomial, Gaussian and Bernoulli. Multinomial Naïve Bayes algorithm has been selected to perform the spam email identification because it is text related and outperforms Gaussian and Bernoulli [22], [23].

Multinomial Naïve Bayes (MNB) classifier uses Multinomial Distribution for each given feature, focusing on term frequency. The Multinomial Naïve Bayes can be denoted as equation-3 [23]:

$$P(p|n) \propto P(p) \prod_{1 \le k \le nd} P(t_k|p) \quad (3)$$

where the number of token is represented by nd, n is the number of emails and $\text{P}(t_k|p)$ is calculated by:

$$\text{P}(\text{t}_\text{k}|\text{p}) = \frac{(\text{count}(\text{t}_\text{k}|\text{p}) + 1)}{(\text{count}(\text{t}_\text{p}) + |\text{V}|)} \quad (4)$$

In the equations (3) and (4), $P(t_k|p)$ is identified as the conditional probability for MNB. The t_k is the spam term occurrence within an email and $P(p)$ is classed as the prior probability. 1 and |V| are identified as the smoothing constant for the algorithm.

To test this algorithm, MNB module was loaded from the Scikit-learn library. The parameters for this model are optional. If none is specified, the default values are: Alpha value set to '1.0', Fit Prior is set to 'True' and Class Prior is set to 'None' [23], [24].

The algorithm-1 shows the pseudocode for Multinomial Naïve Bayes with spam classification where ''Tr'' is Training and ''Te'' is Testing. The $\hat{\text{P}}(t_k|p)$ is the estimating/predicting variable, also known as the conditional probability.

Algorithm 1: Multinomial Naïve Bayes

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
for i = 0; i < TrX; i++ do
    if (i,y) = Spam then
        Learn i = Spam;
    else
        Learn i = Ham;
for t in testSize // Test sizes = 20, 25,
    30 and 40
do
    for K in CV do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        for i = 0; i < TeX; i++ do
            Calculate P̂(t_k|p);
            Calculate the Accuracy;
    return t_k;
```

B. SUPPORT VECTOR MACHINE

This algorithm plots each node from a dataset within a dimensional plane and through classification technique the cluster of data is separated by a hyperplane into their respective groups [25]. The hyperplane can be described as equation-5:

$$H = VX + c \quad (5)$$

where c is a constant and V is the vector. The SGD Classifier was loaded from scikit-learn library, which is the linear model with 'Stochastic Gradient Descent (SGD)', also known as the optimized version of SVM. This algorithm provides more accurate results than SVM (SVC algorithm) itself. Disadvantage of working with SVC algorithm is that it cannot handle a large dataset, whereas SGD provides efficiency and other tuning opportunities.

The algorithm-2 shows the pseudocode for Stochastic Gradient Descent.

The model was implemented with 'Alpha', 'Epsilon' and 'Tol' values with default as 'Hinge' for loss providing linear SVM, also known as 'Soft-Margin' which is easier to compute [25].

The algorithm uses the learning rate to iterate over the sample data to optimize the Linear algorithm and it is denoted by the following equation-6 for the default learning rate as 'Optimal':

$$\frac{1}{\alpha(t_0 + t)} \quad (6)$$

where t is the time step which is acquired by multiplying number of iterations with number of samples (Emails). The Learning Rate allows implementation of the parameter space

Algorithm 2: Stochastic Gradient Descent

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
Initialise Alpha, Epsilon values;
for i = 0; i < TeX; i++ do
    Calculate Hyperplane // Equation (5)
    Measure the Distance (X_i, X_j);
    for t in testSize // Test sizes = 20, 25,
        30 and 40
    do
        for K in CV do
            X_test and y_test z testing size;
            X_train and y_train = training size;
            Call the SGD function;
            Calculate the Training Error;
            Calculate the Rate;
            Calculate the Accuracy;
    return t_k;
```

during the training time. The α is represents the regularization term and t_0 is a hueristic approach.

C. DECISION TREE CLASSIFIER

The Decision Tree model is based on the predictive method. The model creates a category which is further distributed into sub-categories and so on. The algorithm runs until the user has terminated or the program has reached its end decision. The model predicts the value of the data by learning from the provided training data. The longer and deeper the tree implies it has more complicated rules to be executed.

The algorithm-3 shows the pseudo-code for Decision Tree, where it terminates at the end of the node for each split of the tree depth.

Similar to MNB and SGD, Decision Tree (DT) algorithm was loaded from the Scikit-learn library and it is executed on the default parameters which are 'Gini' for Criterion and 'best' for Splitter. The advantage of Gini is that it calculates the incorrectly labelled data that was selected randomly [26]. This is given by the below equation-7:

$$\text{Gini} : G_i = 1 - \sum_{k=1}^{n} p_{(i,k)^2} \tag{7}$$

The second criterion is 'entropy' which is based on information gain based on the selected attributes and it is calculated by equation-8 [26]:.

$$\text{Entropy} : H_i = \sum_{\substack{k=1 \\ p_i k \neq 0}}^{n} p_{(i,k)} \log_2(p_{(i,k)}) \tag{8}$$

where P is the probability and i is a node from the training data within both equation (7) and (8).

Algorithm 3: Decision Tree - CART Algorithm

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
Ln = Number of Leaves;
D = Tree Depth ;
C = Criterion // (G_i) or (H_i)
for t in testSize // Test sizes = 20, 25,
    30 and 40
do
    for K in CV do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        for i < X do
            Call DT function;
            for j < D do
                Calculate the best split;
                Predict the class (c);
                Ln++;
                For the node: (c,C) // Use
                    equation (7) or (8)
                return Predicted Class (ĉ)
            Calculate the Accuracy;
```

D. RANDOM FOREST CLASSIFIER

Random Forest (RF) algorithm can be used for both classification and regression. The algorithm predicts the classes by using multiple decision tree, where each tree predicts the classification class. This is evaluated by the RF model to select the high number of predicted class as an assigned prediction [27].

The algorithm-4 explains the workings of the Random Forest classifier with the Spam Email dataset, where Fĉ is the outcome predicted from the entire forest.

Equation-7 and equation-8 are also utilised to calculate the Gini and Entropy for Random Forest (RF) algorithm to calculate the Criterion.

This module was loaded from Scikit-learn library and it is based on the depth of the tree and number of DT to be produced. These are usually considered as the termination criteria. This means the more the depth and the number of trees the more the computational time required for the algorithm.

E. MULTI-LAYER PERCEPTRON (MLP)

The MLP is a feed-forward Artificial Neural Network (ANN). It is a supervised method which includes non-linear hidden layers between the input and the output layer. The algorithm works with the linear activation function on a training dataset set by default known as Hyperbolic Tan

Algorithm 4: Random Forest

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
Ln = Number of Leaves;
D = Tree Depth ;
C = Criterion // (Gi) or (Hi)
Nt = Number of Trees;
for t in testSize // Test sizes = 20, 25,
    30 and 40
do
    for K in CV do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        for i < Nt do
            for i < D do
                Randomly select from X;
                Split the nodes with C;
                Calculate the Predicted ĉ from each tree;
                return Predicted from the tree Tĉ
            Provide the array of DT decision for all trees;
        Calculate the prediction from the majority;
        return Predicted from the Forest Fĉ
```

(equation-9) [28]:

$$f(\cdot) : R^m \rightarrow R^o \tag{9}$$

where m is the input (spam words in this case) and o is the number of outputs from the function. The algorithm can have one or more layers between input and output layer known as 'Hidden Layer(s)'. The hidden layer accepts the values from the previous layer and transforms with linear summation, whereas the 'Output' layer provides the output values after transformation from the previous hidden layer [28].

The algorithm-5 shows the pseudocode for Multi-Layer Perceptron.

The algorithm uses back-propagation technique to calculate the gradient descent for each variable weight. The algorithm has the ability to learn when it becomes part of one neuron and one hidden layer of MLP function as indicated in equation-10.

$$f(x) = W_{2g}(W_1^T x + b_1) + b_2 \tag{10}$$

where W_2 and W_1 are the weights from the input layer and hidden layer. The W_i^1 becomes the part of n_i layers in the hidden layer [28]. To compare the results of NN and ML models, the modules were loaded from the Scikit-learn similar to the ML models. The default parameter was changed for hidden layer to lower number of neurons for faster computation.

Algorithm 5: Multi-Layer Perceptron

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
H = No.Hidden Layer;
Nu = No.of Neurons;
Ac = Activation;
S = Solver;
for t in testSize // Test sizes = 20, 25,
    30 and 40
do
    for K in CV do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        Call the MLP Function ← H, Nu, Solver ;
        Calculate the Error Rate;
        Error × Activation;
        Calculate the Accuracy;
```

VI. PROGRAM STRUCTURE, DATASETS AND REQUIREMENTS

The Python program will load all the necessary Python libraries that will assist the ML modules to classify the emails and detect the spam emails.

A. ADDING CORPUS

This section will load all the email datasets within the program and distribute into training and testing data. This process will be accepting the datasets in '*.txt' format for individual email (Ham and Spam). This is to help understand the real-world issues and how can they be tackled.

B. TOKENIZATION

Tokenization is the method where the sentences within an email are broken into individual words (tokens). These tokens are saved into an array and used towards the testing data to identify the occurrence of every word in an email. This will help the algorithms in predicting whether the email should be considered as spam or ham [49].

C. FEATURE EXTRACTION AND STOP WORDS

This was used to remove the unnecessary words and characters within each email, and creates a bag of words for the algorithms to compare against.

The module 'Count Vectorizer' from Scikit-learn assigns numbers to each word/token while counting and provides its occurrence within an email. The instance is invoked to exclude the English stopwords, and these are the words such as: A, In, The, Are, As, Is etc., as they are not very useful to classify whether the email is spam or not. This instance is then fitted for the program to learn the vocabulary [49].

Once tokenized, the program applies 'TfidfTransformer' module to compute the Inverse Document Frequency (IDF). The most occurring words within the documents will be assigned values between 0-1, and lower the value of the word means that they are not unique. This allows the algorithms/modules to read the data [49]. The TF-IDF can be calculated by the Equation-11 where (t, d) is the term frequency (t) within a document (d):

$$\text{tf} - \text{idf(t, d)} = \text{tf(t, d)} \times \text{idf(t)} \tag{11}$$

where IDF is calculated by the Equation-12, given n is the number of documents:

$$\text{idf(t)} = \log(\frac{\text{n}}{\text{idf(t)}}) + 1 \tag{12}$$

D. MODEL TRAINING AND TESTING PHASE

As discussed through the research, supervised learning methods were used and the model was trained with known data and tested with unknown data to predict the accuracy and other performance measures. To acquire the reliable results K-Fold cross validation was applied. This method does have its disadvantages such as, there is a chance that the testing data could be all spam emails, or the training set could include the majority of spam emails. This was resolved by Stratified K-fold cross validation, which separates the data while making sure to have a good range of Spam and Ham into the distributed set [50].

Lastly, parameter tuning was conducted with the Scikit-Learn and bio-inspired algorithms approach to try and improve the accuracy of ML models. This provides a platform to compare the Scikit-learn library with the bio-inspired algorithms

E. DATASETS

The project accessed the publicly available datasets and included each email as an individual text file. The text files were string based. A list of the few spam email datasets from the public repository are explained below:

1) Ling-Spam dataset is divided into 10 parts from the 'bare' distribution that includes individual emails as a text file (.txt). This data is not pre-processed, and it includes numbers, alphabets and characters [29]. Each part was trained and tested to acquire the average accuracy.
2) Enron dataset includes 6 separate datasets that contain 3000-4000 individual emails as text files. The dataset includes numbers, alphabets and characters [30].
3) The PUA dataset is a numerical dataset that includes sets/combination of numbers characterised as a string. PU1, PU2 and PU3 are similar to PUA dataset but include different weights of spam and ham emails and they are extracted from different users [31]. Folders include individual emails as a text file. For all PU datasets, the publisher has replaced the tokens with a unique combination of numbers to respect their user's privacy. The respective words for these unique numbers have not been made public, making the process of removing certain features difficult [32].
   From PU1 and PU2 datasets, duplicate emails that were received were discarded manually. Whereas in PU3 and PUA, this was conducted with the UNIX command 'diff'. Each of these emails were collected during different lengths of time for both Ham and Spam emails.
4) SpamAssassin dataset is more advanced with header information such as source or From address, IP address, return path, message ID and delivery information. Each individual email within the folder will be converted into text files [33].

Table-2 presents the spam rate of each of the datasets that are used within this project along with their published date [2].

TABLE 2. Datasets.

Name of the Dataset	Ref.	Spam + Ham = Total emails	Rate of Spam	Published Date
Ling-Spam	[29]	481+2412 = 2893	17%	2000
PU1	[31]	481+618 = 1099	44%	2000
SpamAssassin	[33]	1897+4150 = 6047	31%	2002
PUA	[31]	571+571 = 1142	50%	2003
PU2	[31]	142+579 = 721	20%	2003
PU3	[31]	1826+2313 = 4139	44%	2003
Enron 1 – 6 Spam	[30]	20170+16545 = 36,715	55%	2006

F. SOFTWARE AND HARDWARE

Python 3.4 or above was used and Anaconda platform seemed like a good option as it provides the advantage of using both Spyder and Jupyter Notebook for implementing the programs.

Some online applications such as Google Collaboratory and Kaggle can be used to speed up the training and testing process for the multiple datasets. This can be helpful towards any NN algorithms that can be implemented.

The project was conducted on a standard laptop, with 8 GB RAM and AMD Ryzen 3 3200U (2.60 GHz) processor.

G. LIBRARIES AND MODULES

Scikit-Learn will be used as it offers the majority of the machine learning libraries and dataset processing modules.

As per the papers discussed in the related work section, PSO performed much better among the bio-inspired algorithms. For comparison purposes, the second implementation of bio-inspired algorithm will be based on human evolution. Libraries like PySwarms for Particle Swarm Optimisation and TPOT for Genetic Algorithm will be utilised to optimise the accuracy of the machine learning algorithms.

VII. PERFORMANCE MEASURES

There are different performance metrics that were used in this work as follows.

	HAM	SPAM
HAM	TN	FP
SPAM	FN	TP

A. CONFUSION MATRIX

The detection of spam emails can be evaluated by different performance measures. Confusion Matrix is being used to visualise the detection of the emails for models. Confusion matrix can be defined as below:

where [34]:

1) TN = True Negative – Ham email predicted as ham
2) TP = True Positive – Spam email predicted as spam
3) FP = False Positive – Spam email predicted as ham
4) FN = False Negative – Ham email predicted as spam

B. ACCURACY

The research was aimed at finding the highest accuracy for detecting the emails correctly as ham and spam. The module from the Scikit-learn library called 'Accuracy' helped analyse the correct number of emails classified as 'Spam' and 'Ham'. This can be measured by equation-13 below [35]:

$$\text{Accuracy} = \frac{(\text{TN} + \text{TP})}{(\text{TP} + \text{FN} + \text{FP} + \text{TN})} \tag{13}$$

where the denominator of the equation is the total number of emails within the testing data.

C. RECALL

The recall measurement provides the calculation of how many emails were correctly predicted as spam from the total number of spam emails that were provided. This is defined by equation-14, where 'TP + FN' are the total number of spam emails within the testing data [35]:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{14}$$

D. PRECISION

The precision measurement is to calculate the correctly identified values, meaning how many correctly identified spam emails have been classified from the given set of positive emails. This means to calculate the total number of emails which were correctly predicted as positive from amongst the total number of emails predicted positive [35]. This is defined by equation-15:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{15}$$

E. F1-SCORE

The F-measure or the value of F_β is calculated with the help of precision and recall scores, where β is identified as 1, F_β or F_1 provides the F1-score. F1-score is the 'Harmonic mean' of the precision and recall values. This can be defined by equation-16 [35]:

$$F_\beta = \frac{(1 + \beta^2)(\text{precision} \times \text{recall})}{(\beta^2 \times (\text{precision} + \text{recall}))} \tag{16}$$

when the β is substituted with the value 1, the formula is simplified to:

$$F_\beta = \frac{2 \times (\text{precision} \times \text{recall})}{(1 \times \text{precision} + \text{recall})} \tag{17}$$

VIII. RESULTS OF BASE MODEL ON DATASETS

Stratified K-Fold Cross Validation (SKFCV) was applied to all the machine learning models. Algorithm-6 represents the pseudocode for the base models, this was executed with the default values for parameters explained in section V and VI.

Algorithm 6: Base Model Implementation With SKFCV

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
X_i = StopwordRemoval
X_j = Vectorizer
X_k = Tf-IDF
Dict ← X_k
Dict = Pre-Processed data;
Initialise SKF // Stratified K-Fold Cross
    Validation
for t in testSize // Test sizes = 20, 25,
    30 and 40
do
    for K in SKF do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        Calculate the Accuracy;
```

The visual representation of five models is shown in Figure-2: ML algorithms, from left: SGD, MNB, RF, DT and MLP. The figure displays the 4 split sets used for each classifier (CLF). The selected algorithms have provided 90% and above accuracy for email detection except RF. This was applied on the 7 datasets and the average was taken. Amongst the five algorithm RF has performed poorly and SGD is the highest performing algorithm.

The respective accuracies for each split set in Figure-2 are defined in Table-3.

Computational timing depends on the depth of a dataset and the classification. For base classifiers, the approximate times are shown below to train for each iteration of cross-validation for the respective classifiers.

1) Naïve Bayes: 0.003 sec to 0.0013 sec approx.
2) Support Vector Machine: 0.040 sec approx.
3) Random Forest: 1.080 sec approx.
4) Decision Tree: 4.06 sec approx.
5) Multi-Layer Perceptron: 8.0 sec approx.

The experiment evaluates that the more the training data, the better accuracy the testing data provides. The NN model will later be tested for 75:25 and 80:20 split set with bio-inspired algorithms.

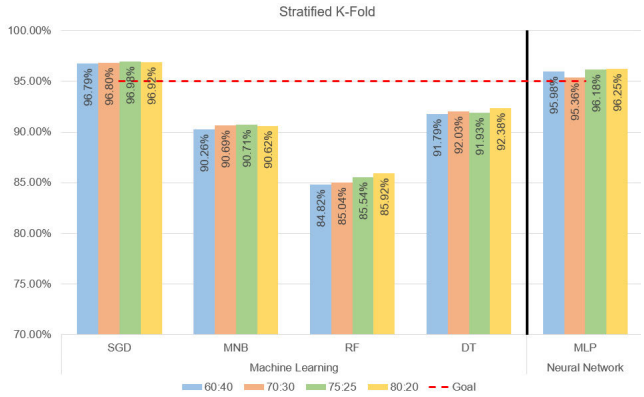


FIGURE 2. Stratified K-fold cross validation.

TABLE 3. Stratified K-fold cross-validation - accuracy.

Classifier	Split Set			
	60:40	70:30	75:25	80:20
SGD	96.79%	96.80%	96.98%	96.92%
MNB	90.26%	90.69%	90.71%	90.62%
RF	84.82%	85.04%	85.54%	85.92%
DT	91.79%	92.03%	91.93%	92.38%
MLP	95.98%	95.36%	96.18%	96.25%

IX. TUNING OF PARAMETERS

For every model, certain parameters were selected and provided with a range of possibilities. These parameters are the ones that have high impact towards detecting the emails and learning rate. This will then be implemented within bio-inspired algorithms.

A. SGD PARAMETERS

Hyperparameter tuning the algorithm offers 3 parameters from the SGD algorithm: Alpha values, Epsilon values and Tol values for the search space. Values for all three keys ranged from 0.0001 to 1000 as a dictionary.

- Alpha: The variable could help set the optimal learning rate. It is also classed as the constant for regularization term.
- Epsilon: This value determines the learning rate for the algorithm.
- Tol: This is the criteria for termination.

B. MNB PARAMETERS

The dictionary of parameters provided for the optimization were values of:

- Alpha: This is used as a smoothing parameter for Laplace or Lidstone to the raw counts. This parameter will be passed as a float for PSO. This is combined with the number of features within the module. The value ranged from 0.0001 to 1000 as a dictionary
- Fit Prior: This is to learn the class probabilities.

C. RF PARAMETERS

The dictionary of parameters provided for the optimization were values of:

- Number of estimators: This states number of trees in the forest.
- Max depth: This indicated maximum depth of the tree.
- Minimum leaf sample: Specifies the minimum number of leaves at the leaf node.
- Criterion: This is in a string format. This is a tree specific parameter that can be 'Gini' for Gini impurity or 'Entropy' for information gain.

D. DT PARAMETERS

The dictionary of parameters provided for the optimization were values of:

- Splitter: This is a string-based parameter which can be either 'Best' or 'Random'. This specifies the strategy for the split at a node.
- Max Depth: This will specify the depth of the tree.
- Criterion: This measures the quality of the split.
- Minimum leaf sample: This is passed as an integer to specify the minimum number of samples that are necessary at the leaf node.

E. MLP PARAMETERS

The dictionary of parameters provided for the optimization were values of:

- Hidden layer sizes: Number of neurons to be considered by the classifier. This is where each feature is interconnected with each neuron.
- Alpha: This is a regularization parameter. The value was ranged from 0.001 to 0.01. These values were less than the default value.
- Solver: According to the Scikit-learn documentation, the solver when set to 'LBFGS', the module's performance and speed can increase on small datasets.

Due to the computational time required for the MLP classifier, for this project purpose, the optimisation was done on '5' hidden layers and the solver set to 'LBFGS' which is an optimizer that can converge fast and provide better performance. The greater number of neurons added to the hidden layer, the more time it will require to train the model [28].

X. BIO-INSPIRED OPTIMIZATION ALGORITHMS

There are two bio-inspired optimization approaches that are discussed here which helped to improve the results of the experiments, i.e. Particle Swarm Optimisation and Genetic Algorithm.

A. PARTICLE SWARM OPTIMISATION

The PSO is based on the swarming methods observed in fish or birds. The particles are evaluated based on their best position and overall global position. Particles within a search space are scattered to find the global best position.

The Pyswarms library offers different calculations and techniques for PSO to be used with an ML model such as feature subset selection or parameter tuning optimization. As researched in the previous sections, the feature selection can reduce feature space but can also discard some features that can be useful during the classification. Therefore, PSO will be used to tune and find the hyper-parameter for a given ML/NN model.

The PSO will use the 'GlobalBestPSO' from the 'pyswarms.single.global_best' module. This will then use the 'optimize' method with an objective function and number of iterations to run the PSO before terminating. This will then provide the 'Global best cost' and 'Global Best position'' [36].

The 'global_best' module and equation-18 denotes the updating of each particle position:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{18}$$

where x_i is the position of the particle, t is the current timestamp and $t+1$ is the computed velocity which gets updated. The velocity (v_i) can be further defined as below:

$$v_{ij}(t+1) = w \times v_{ij}(t) + c_1 r_1 j(t)[y_{ij}(t) - x_{ij}(t)] + c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \tag{19}$$

where r_1 and r_2 are the random numbers, c_1 is the cognitive parameter and c_2 is the social parameter. These parameters control the behaviours of the particle. The w is the inertia parameter that controls the swarm's movements, which is the important parameter and hence the value is bigger than the other parameters. The parameters for cognitive and inertia parameter remained with default value as the demonstrated algorithm in 'Optimizer' package [36].

The social parameter was increased by 0.5. The parameters passed onto the Global_best module are:

- Number of particles: 10; this was considered by the examples set within the Pyswarms library.
- Dimension: This is the number of dimensions within a given space. The number of parameters for the base algorithms such as Alpha, Tol, Epsilon etc.
- Options: C1 = 0.5, C2 = 0.7 and W = 0.9. These parameters have effect on the computation time.
- Bounds: This is a tuple, obtained through the dimension. Higher and lower value within the base algorithm's parameters will be considered.

The option setting of coefficients is important. The smaller the number, the distance of the particle movement will be small too. This can take more time in computing the models.

Algorithm-7 shows the pseudocode for Particle Swarm Optimization. This is implemented on top of the base model in algorithm-6.

Figure-3 shows the visual representation of PSO algorithm accuracies for the 5 models/classifiers. The accuracy score was taken from the average of all seven datasets. The highest accuracy of 98.47% was provided by Naïve Bayes on 80% training data and 20% testing data. Overall, MNB provided higher accuracy from all the other classifiers.

The respective accuracies for each split set in the above figure-3 are defined in Table-4.

The entire program took nearly a day for all 5 classifiers to run on different platforms. The computational

Algorithm 7: PSO Implementation

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
X_i = StopwordRemoval;
X_j = Vectorizer;
X_k = Tf-IDF;
Dict ← X_k1
Dict = Pre-Processed data;
Initialise ML Parameters; // This will include
    the key and the values
Declare ML Algorithm; // MNB, SGD, DT,
    RF and MLP
Def PSO:
    Initialise PSO parameters;
    C1 = 0.5; // Cognitive Parameter
    C2 = 0.7; // Social Parameter
    W = 0.9; // Weight
    N_i = NumberOfIteration;
    N_p = NumberOfParticles;
    Calculating the Dimension;
    (Key,Value) ← Parameters; // The
        parameters of the algorithm i.e
        MNB, SGD
    Call PSO G_Best algorithm; // Global Best
    PSO Module ← Dimension, C1, C2, w, N_i, N_p;
    Call Objective Function O_f;
    PSO ← O_f;
    Calculate the Best_Position of the Swarm;
    Best_Position ← N_i;
    Calculate the Measures;
    Measures ← Best_position, TrData, TeData;
    return Accuracy
Def O_f:
    for i < N_p do
        Initialise StratifiedKF;
        Calculate the Score;
        return The array of accuracies A_q
            // conducted with the
            dimensions and the Key and
            Value provided
    for t in test size do
        X_test and y_test = testing size;
        X_train and y_train = training size;
        Call the function PSO (training and testing data);
```

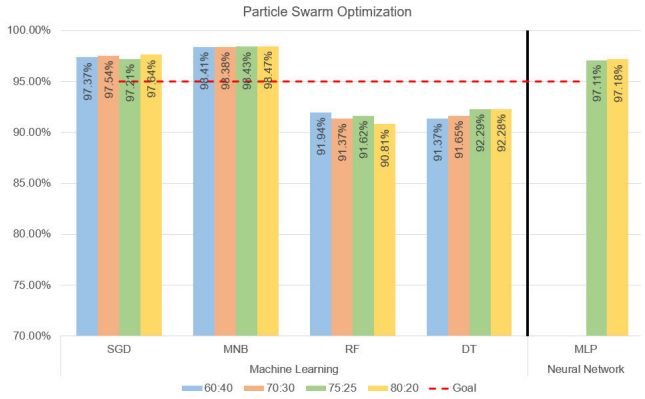


FIGURE 3. Particle swarm optimization – accuracy.

TABLE 4. PSO - accuracy.

Classifier	Split Set			
	60:40	70:30	75:25	80:20
SGD	97.37%	97.54%	97.21%	97.64%
MNB	98.41%	98.38%	98.43%	98.47%
RF	91.94%	91.37%	91.62%	90.81%
DT	91.37%	91.65%	92.29%	92.28%
MLP	-	-	97.11%	97.18%

time required for the runs depending on the dataset are as follows:

1) Multinominal Naïve Bayes: 5 mins approx.
2) Support Vector Machine: 5 mins approx.
3) Random Forest: 2 mins to 15 mins approx.
4) Decision Tree: 2 mins to 25 mins approx.
5) Multi-Layer Perceptron: 25 min to 1hour approx.

In terms of datasets, the highest achieving algorithm is MNB with 70:30 split set on SpamAssassin dataset. The parameters chosen were: Alpha: '0.0004940843999793119' and Fit Prior: 'false'.

The highest occurred accuracy from the given datasets along with the classifier (CLF), Test Size and the parameters that were selected by the PSO algorithm is shown in table-5.

Tables 6, 7, 8 and 9, represent the F1-score, Precision and Recall in comparison to Accuracy. It shows the average of performance measurements for the ML algorithms applied on 7 datasets. The MNB algorithm was the one which provided the best performance amongst other ML algorithms for all four different split sets. The percentage calculated are taken from the average of all 7 datasets.

From these tables, 98% of the emails were correctly detected by MNB on the average. The average precision was 97.50% and average recall was 97.40% and average F1-score was 97.50%.

The highest accuracy noted was 98.47% achieved by MNB, providing precision of 97.23%, recall of 97.86% and F1-Score of 97.54%. This was achieved with training size 80% and Testing size 20%.

TABLE 5. PSO selected values.

CLF	Dataset	Test Size	Acc.	Parameters
MNB	SpamAssassin	70:30	99.89%	Alpha: 0.0004940843999793119, Fit Prior: false
SGD	SpamAssassin	80:20	99.67%	Alpha: 0.00011028738827772605, Epsilon: 0.0008410556148690041, Tol: 0.0004946026859321408
RF	SpamAssassin	60:40	97.75%	No. of estimators: 5, Max Depth: 7.154304302293813, Min leaf sample: 1, Criterion: 'entropy'
DT	SpamAssassin	80:20	99.33%	Splitter: 'best', Criterion: 'entropy', Max Depth: 6.910921197436718, Min Leaf Sample: 2
MLP	SpamAssassin	80:20	99.67%	Hidden Layer Size: (5,), Alpha: 0.005177077568305584, Solver: 'lbfgs'

TABLE 6. PSO 60:40 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.37%	95.37%	97.18%	93.83%
MNB	98.41%	97.80%	97.55%	97.55%
RF	91.94%	79.57%	94.23%	74.76%
DT	91.37%	85.16%	88.10%	83.24%

TABLE 7. PSO 70:30 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.54%	96.21%	97.24%	95.20%
MNB	98.38%	97.51%	97.82%	97.31%
RF	91.37%	77.67%	96.95%	70.36%
DT	91.65%	86.64%	87.81%	85.96%

TABLE 8. PSO 75:25 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.21%	94.79%	97.11%	93.11%
MNB	98.43%	97.60%	97.73%	97.24%
RF	91.62%	77.88%	96.58%	92.51%
DT	92.29%	87.48%	88.27%	88.04%

TABLE 9. PSO 80:20 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.64%	95.78%	96.80%	95.59%
MNB	98.47%	97.54%	97.23%	97.86%
RF	90.81%	74.79%	96.11%	66.49%
DT	92.28%	86.71%	88.07%	86.45%

B. GENETIC ALGORITHM

The GA algorithm is an evolutionary algorithm based on Darwinian natural selection that selects the fittest individual from the given population. This involves the principle of variation, inheritance and selection. The algorithm maintains a population size and the individuals have a unique number

(Chromosomes) that are binary represented. The algorithm iterates through a fitness function where best individuals are selected for reproduction of the offspring. The higher the fitness, the higher the probability [8].

Implementation of the GA was conducted with the help of TPOT library. The program selects the best parameters from a given dictionary of parameters. The TPOT classifier is then trained with cross validation. The parameters given to the TPOT are as follows [37]:

- Generation: Number of times the pipeline will conduct the optimization process. The default value is 100. The program has set this parameter as '10'.
- Population size: Number of individuals participating for Genetic programming within each generation. Default is 100. The program has set this parameter as '40'
- Offspring size: Offspring to be produced in each generation. Default is 100. The program has set this parameter as '20'.

The program runs for 10 generations with 40 population size and 20 offspring production. This means 400 (10×40) hyperparameter combinations will be evaluated before terminating for each generation. Each pipeline will be evaluated with 10-fold cross validation i.e. 400×10. Once the TPOT classifier is terminated, it provides the best pipeline parameters. The entire pipeline will be evaluated [(Generation x lambda) + Population size] = 240 times, where lambda is Offspring size. If no Offspring size is provided the pipeline will evaluate by substituting population as 'lambda'.

The mutation rate and the crossover rate were set as default. The mutation rate is 0.9, which is the changes in the parameter value. The crossover rate is 0.1, which is the percentage of the individuals required from the population to create offspring. The TPOT warns that 'Mutation rate + crossover rate' should not exceed 1.0.

Algorithm-8 shows the pseudocode for Genetic Algorithm. This is implemented on top of the base model in algorithm-6.

The Figure-4 shows the visual representation of GA algorithm accuracies for the 5 classifiers.

The respective accuracies for each split set in the figure-4 are defined in Table-10.

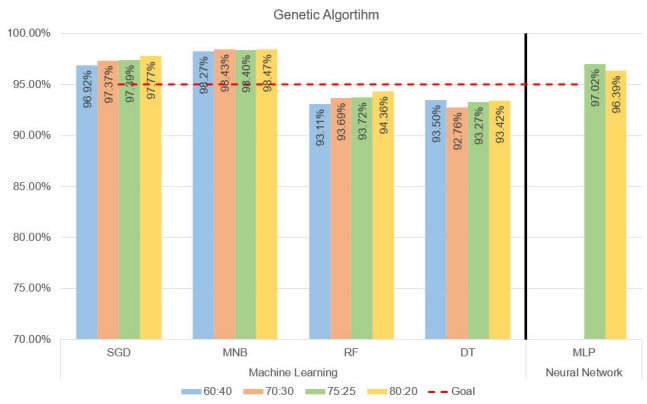


FIGURE 4. Genetic algorithm – accuracy.

Algorithm 8: GA Implementation

```
Initialise Input Variables;
N ← No. of Documents;
X ← Datapoints;
y ← Target Inputs;
X_i = StopwordRemoval;
X_j = Vectorizer;
X_k = Tf-IDF;
Dict ← X_k;
Dict = Pre-Processed data;
Initialise ML Parameters; // This will include
    the key and the values
Declare ML Algorithm; // MNB, SGD, DT,
    RF and MLP
Def GA:
    Initialise GA parameters;
    G = GenerationSize;
    P = PopulationSize;
    O_s = OffSpringSize;
    M = MutationRate;
    C = CrossoverRate;
    K = StratifiedKF; // Cross Validation
    GA(TPOT) Module ← G, P, O_s, M, C;
    Calculate the Survivor of the Swarm;
    for G in Generation do
        for P in Population do
            for i < K do
                Survival ← Calculate the Fitness;
                Select two Individual;
                Produce OffSpring ← O_s;
                Mutate ← OffSpring, M;
                return KScore
            Calculate Parameters;
        return Parameters
    Calculate the Measures;
    Measures ← Parameters, TrData, TeData;
    return Accuracy
for t in test size do
    X_test and y_test = testing size;
    X_train and y_train = training size;
    Call the function GA (training and testing data);
```

The computational time required for the runs depending on the dataset is given as:

TABLE 10. GA - accuracy.

Classifier	Split Set			
	60:40	70:30	75:25	80:20
SGD	96.92%	97.37%	97.39%	97.77%
MNB	98.27%	98.43%	98.40%	98.47%
RF	93.11%	93.69%	93.72%	94.36%
DT	93.50%	92.76%	93.27%	93.42%
MLP	-	-	97.02%	96.39%

1) Mutinomial Naïve Bayes: 12 mins approx.
2) Support Vector Machine: 15 mins approx.
3) Random Forest: 3 mins to 20 mins approx.
4) Decision Tree: 15 mins to 40 mins approx.
5) Multi-Layer Perceptron: 1 hour to 2 hours approx.

Table-11 shows the output for every classifier that achieved highest accuracy, which is similar to that conducted with PSO. The highest achieving accuracy of 100% was by MNB on 80:20 split set with SpamAssassin dataset. The parameters chosen were: Alpha: 0.01, Fit Prior: 'false'.

TABLE 11. GA selected values.

CLF	Dataset	Test Size	Acc.	Parameters
MNB	SpamAssassin	80:20	100%	Alpha: 0.01, Fit Prior: 'false'
SGD	SpamAssassin	80:20	99.83%	Alpha: 0.0001, Epsilon: 1.0, Tol: 0.1
RF	SpamAssassin	75:25	98.54%	Criterion: 'entropy', Max Depth: 30, Min Leaf Sample: 1, No. of estimators: 25
DT	SpamAssassin	80:20	99.33%	Criterion: 'entropy', Max Depth: 15, Min Leaf Sample: 1, Splitter: 'best'.
MLP	SpamAssassin	75:25 and 80:20	99.33%	Alpha: 0.001, Solver: 'lbfgs'.

Tables 12, 13, 14 and 15, represent the F1-score, Precision and Recall in comparison to Accuracy when the Genetic Algorithm (GA) is applied on the machine learning (ML) algorithms. It shows the performance measurements for the ML algorithms. The MNB algorithm was the one to provides the best performance amongst other ML algorithms for all four different split sets like it did with PSO. The percentages shown are calculated from the average of all 7 datasets.

TABLE 12. GA 60:40 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	96.92%	95.27%	96.59%	94.13%
MNB	98.27%	97.32%	97.38%	84.63%
RF	93.11%	83.13%	96.51%	77.16%
DT	93.50%	88.42%	90.02%	86.96%

TABLE 13. GA 70:30 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.37%	95.61%	96.98%	94.52%
MNB	98.43%	97.64%	97.76%	97.61%
RF	93.69%	85.83%	97.00%	80.11%
DT	92.76%	88.25%	89.34%	87.48%

TABLE 14. GA 75:25 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.39%	95.68%	97.48%	94.03%
MNB	98.40%	97.57%	98.09%	97.11%
RF	93.72%	85.73%	97.25%	80.43%
DT	93.27%	88.72%	90.68%	87.55%

TABLE 15. GA 80:20 split set.

Classifier	Accuracy	F1-score	Precision	Recall
SGD	97.77%	96.71%	97.61%	95.97%
MNB	98.47%	97.67%	98.01%	97.59%
RF	94.36%	87.42%	97.79%	81.74%
DT	93.42%	89.54%	91.07%	88.51%

From these tables, 98% of the emails were correctly detected by MNB on the average. The average precision was 97.50%, average recall was 93.00% and average F1-score was 97.00%.

The highest accuracy was 98.47% achieved by MNB, providing precision of 97.79%, recall of 81.74%, and F1-Score of 87.42%. This was achieved with training size 80% and Testing size 20%.

XI. RESULTS OF OPTIMIZED CLASSIFIERS ON DATASETS

There are two types of spam email dataset being used for this project, alphabetic-based and numeric-based files. These are described in table-16.

TABLE 16. Dataset type.

Sr.No	Alphabetical Based	Numerical Based
1)	Ling-Spam	PU1
2)	Enron Spam	PU2
3)	SpamAssassin	PU3
4)	-	PUA

Each of the four machine learning models/classifiers were tested with the average taken from the alphabetical datasets and compared with the average taken from the numerical datasets.

Figure-5 shows the split between the two types of dataset, namely numerical and alphabetical. The algorithm SGD provided the highest accuracy for alphabet-based datasets. Even though the accuracies for the numerical datasets are low, the improvement is much better than the base algorithm compared to the alphabet-based dataset.

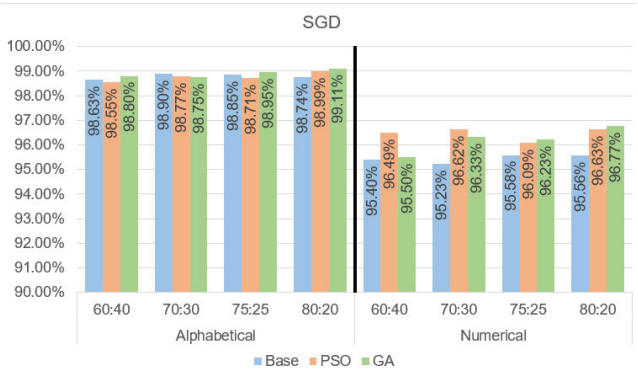


FIGURE 5. Stochastic gradient descent alpha/num comparison.

Figure-6 shows the performance of the MNB algorithm with both type of datasets. The algorithm performed similarly to SGD. The accuracy is higher for the alphabet-based dataset than the numerical dataset.

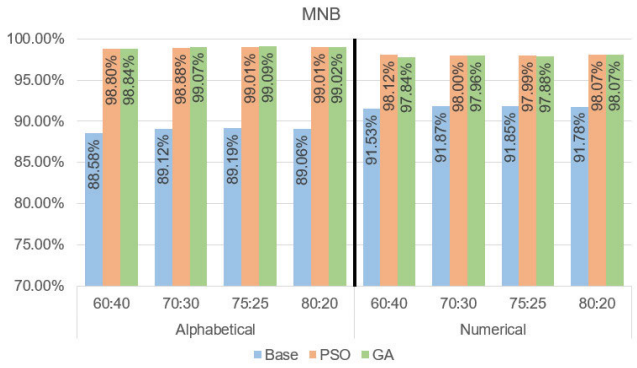


FIGURE 6. Multinomial Naïve bayes alpha/num comparison.

Both MNB and SGD algorithms worked well for numerical and alphabet-based datasets with PSO and GA optimization. The accuracy is higher on the alphabet-based datasets for both algorithms. Split set 75:25 and 80:20 have worked better than the split set 60:40 and 70:30.

Figure-7 shows the performance of RF algorithm between the numerical and alphabetical datasets. This tree-based algorithm seems to have worked well with the numerical datasets in terms of accuracy and improvement.

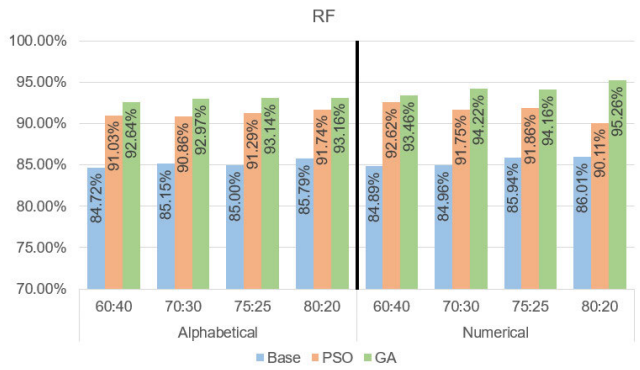


FIGURE 7. Random forest alpha/num comparison.

Figure-8 shows the performance of DT algorithm. Similar to RF, the DT has worked better with numerical in terms of improvement. Whereas for alphabetical, there is very less improvement but higher accuracy.

The tree-based algorithms (Figure-7 and Figure-8) have performed better with GA optimization than the PSO on both type of datasets.

For Neural Networks, the implementation with the PSO algorithm took more than 7 hours for 5/25 iterations to be completed for one split set with 100 neurons. Since the

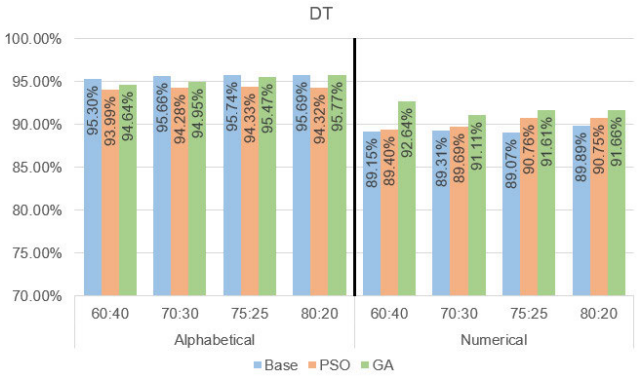


FIGURE 8. Decision tree alpha/num comparison.

MLP classifier was taking more power, the algorithm was distributed between three platforms 1) Kaggle, 2) Google Collaboratory and 3) standard PC. The number of neurons were reduced to '50' to acquire an idea of timing for the run. This time the algorithm took about a minimum of 6 hours to complete one split set. Hence, at the end the classifier was run with 5 Neurons providing some improvement and quicker completion.

The MLP classifier was experimented with the optimization techniques by integrating with PSO and GA. The classifier was testing with 75:25 and 80:20 split sets, as these were the highest providing accuracies for ML models/classifiers. The alphabet-based dataset performed much better than the Numerical dataset as shown in Figure-9.

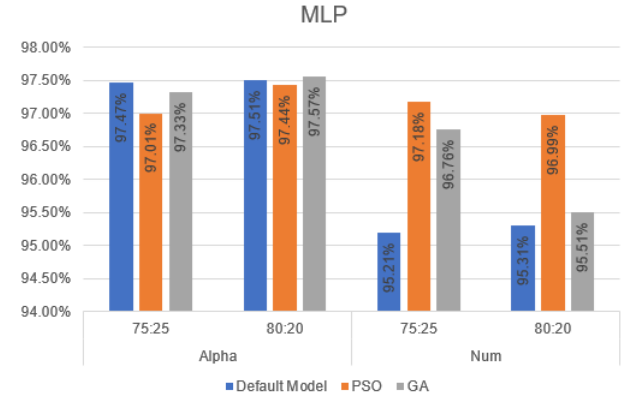


FIGURE 9. Multi-layer perceptron alpha/num comparison.

XII. EVALUATION AND COMPARISON

The evaluation and performance comparison on the work is discussed in this section.

A. SPLIT SETS

Evaluating all the split sets for training and testing data on all seven datasets, sizes 72:25 and 80:20 were the top two splits to provide better accuracy and showed improvement. This could vary on the dataset size and the information separated

TABLE 17. SpamAssassin dataset.

Classifier	Base	PSO	GA	GSCV	RSCV
SGD	99.28%	99.67%	99.83%	99.50%	99.66%
MNB	89.28%	99.83%	100.00%	99.66%	99.66%
RF	89.25%	97.33%	98.17%	98.33%	96.17%
DT	98.47%	99.33%	99.33%	98.83%	99.00%
MLP	99.35%	99.67%	99.33%	-	-

TABLE 18. Enron spam dataset.

Classifier	Base	PSO	GA	GSCV	RSCV
SGD	99.12%	99.20%	99.21%	99.17%	99.14%
MNB	93.26%	98.58%	98.60%	99.52%	98.52%
RF	80.88%	88.45%	94.05%	94.76%	94.27%
DT	96.05%	93.62%	96.07%	95.50%	94.93%
MLP	98.12%	99.18%	99.05%	-	-

TABLE 19. Ling-spam dataset.

Classifier	Base	PSO	GA	GSCV	RSCV
SGD	97.82%	98.11%	98.28%	98.79%	98.79%
MNB	84.65%	98.63%	98.45%	99.48%	98.79%
RF	87.22%	89.45%	87.26%	92.59%	89.84%
DT	92.56%	90.01%	91.90%	92.25%	93.63%
MLP	95.06%	93.46%	94.32%	-	-

TABLE 20. PU3.

Classifier	Base	PSO	GA	GSCV	RSCV
SGD	95.60%	96.73%	96.37%	97.94%	96.61%
MNB	97.87%	97.94%	97.94%	99.03%	98.42%
RF	81.76%	85.17%	96.37%	95.64%	95.03%
DT	92.42%	92.13%	92.74%	91.76%	92.00%
MLP	97.42%	97.22%	97.46%	-	-

during the split. This proves that the higher the rate of training data than testing data, better the performance achieved. This is a good sign, since when considered as a real-world example, the models will have bigger weight for training data than testing.

B. EXPERIMENTAL RESULTS - ACCURACY

According to the experiments, PSO and GA have improved the accuracy of all five models. The Multinomial Naïve Bayes (MNB) is the algorithm that has performed better than all the other algorithms. Comparing across the different types of datasets, Enron, SpamAssassin and Ling-Spam dataset provided more depth by eliminating certain features through the emails, hence, allowing the optimization techniques more search space. But the numerical dataset (PU1, PU2, PU3 and PUA) were very restricted, even though they successfully provided accuracy improvements on some split sets.

Taking the individual datasets into account, the SpamAssassin dataset performed very well with Naïve Bayes and Genetic Algorithm. Table-17 shows the accuracy comparison for SpamAssassin dataset on Machine Learning models for 80:20 split set. The table also compares with the optimization models that is provided by the Scikit-learn library. Grid Search CV (GSCV) and Random Search CV (RSCV) were both implemented within the base model and were loaded from the Scikit-learn library.

In comparison to alphabetical datasets and numerical datasets, the tables 18, 19 and 20 show the accuracy achieved for 80:20 split set.

Taking Enron Spam dataset into account, it was the second-best corpus to work with followed by Ling-Spam.

An average of all four PU datasets were taken into consideration and PU3 dataset provided better results and the highest accuracy amongst all four and that is shown in table 20.

Even though computational cost is low for PSO providing quick results than GA, GA has provided better results for some ML algorithms. The PSO had very less parameters to be considered for each algorithm i.e C1, C2 and W, whereas GA initiated the mutation and crossover of the original population. The MNB performed better once it was tuned automatically by bio-inspired algorithms and it predicts very highly with text-based datasets as it uses the feature vectors. Hence MNB with GA achieved good results overall for the spam datasets.

Table-21 shows the comparison of this work with similar work of other researchers. The table includes 15 additional papers similar to our paper. Some of the research work have defined additional measurements with accuracy. Majority of our work when compared to the others, provided either better accuracy or similar scores. The table displays the highest accuracies based on the datasets for our work and this is presented at the bottom of the table.

XIII. FUTURE IMPLEMENTATION AND RECOMMENDATION

We plan to further carry out the machine learning algorithms to optimize and compare with different bio-inspired algorithms such as Firefly, Bee Colony and Ant Colony Optimization as researched in the previous sections. We could also explore the Deep learning Neural Network with PSO and GA by exploring different libraries such as TensorFlow's DNN Classifier or similar.

We found that the Neural Network algorithm could have worked better with more dimension like providing broader range of values for learning rate, activation, solver, and alpha. If this project is taken further, implementation for MLP could be done through Keras or TensorFlow with GPU application. This will allow the user to input other parameters and a range of possibilities as their key values.

The user can consider implementing the PSO objective Function with RSCV to compare the difference for accuracy improvement. The PSO and GA can provide better accuracies by incorporating NLP techniques.

TABLE 21. Comparison of our work with other works.

Author Name	Dataset Used	Classifier / Optimization	Performance achieved
[3] Awad (2011)	SpamAssassin	Naïve Bayes	Accuracy = 99.46%, Precision = 99.66%, Recall = 98.46%
[4] Mohammed, et al. (2013)	Email-1431	Naïve Bayes	Accuracy = 85.96%
[6] Agarwal & Kumar (2018)	Ling-Spam	PSO – Naïve Bayes	Accuracy = 95.50%, Precision = 96.42%, Recall = 94.50%, F-measure = 95.45%
[8] Taloba & Ismail (2019)	Enron	GA - DT	Accuracy = 95.50%, Precision = 95.50%, Recall = 97.20%, F-measure = 96.30%
[38] Shams & Mercer (2013)	Ling-Spam	Bagged RF	Accuracy = 95.56%
[39] Kumareson (2016)	Ling-Spam	GA-SVM	Accuracy = 94.69%, Precision = 98.52%, Recall = 20.12%
[39] Kumareson (2016)	Enron	GA-SVM	Accuracy = 93.65%, Precision = 96.24%, Recall = 23.54%
[39] Kumareson (2016)	Spam Assassin	GA-SVM	Accuracy = 94.55%, Precision = 99.65%, Recall = 21.98%
[39] Kumareson (2016)	PU1	GA-SVM	Accuracy = 96.25%, Precision = 97.02%, Recall = 18.78%
[40] Faris, et al. (2016)	Spam Assassin	PSO - RF	Accuracy = 97.92%
[41] Temitayo et.al (2012)	Spam Assassin	GA-SVM	Accuracy = 93.50%
[42] Alghoul et al. (2018)	-	ANN with Feed-Forward Backpropagation	Accuracy = 85.31%
[43] Rathi & Pareek (2013)	SpamBase	Random Tree	Accuracy = 99.72%
[44] Gomes et al. (2017)	Enron Dataset	Hidden Markov Model	Accuracy = 91.28%
[45] Yasin & Abuhasan (2016)	-	J48, Bayes Net, SVM, MLP and Random Forest.	Accuracy = 99.10%
[46] Yüksel et al. (2017)	Custom	SVM	Accuracy = 97.60%
[47] Sharma et al. (2013)	SpamBase	Random Committee	Accuracy = 94.28%
[48] Akinyelu et al. (2014)	SpamAssassin and Phishing Corpus	Random Forest	Accuracy = 99.70%
Our work (Gibson, et al.)	Ling-Spam	GA-SGD	Accuracy = 98.77%, Precision = 100.00%, Recall = 94.21%
Our work (Gibson, et al.)	Enron	GA-SGD	Accuracy = 99.21%, Precision = 98.68%, Recall = 99.54%
Our work (Gibson, et al.)	SpamAssassin	GA-MNB	Accuracy = 100.00%, Precision = 100.00%, Recall = 100.00%
Our work (Gibson, et al.)	PU1	GA-MNB	Accuracy = 99.08%, Precision = 99.31%, Recall = 98.63%
Our work (Gibson, et al.)	PU2	GA-MNB	Accuracy = 97.89%, Precision = 90.62%, Recall = 100%
Our work (Gibson, et al.)	PU3	GA-MNB	Accuracy = 97.04%, Precision = 98.61%, Recall = 96.74%
Our work (Gibson, et al.)	PUA	GA-MNB	Accuracy = 97.81%, Precision = 97.76%, Recall = 96.46%

XIV. CONCLUSION

The project successfully implemented models combined with bio-inspired algorithms. The spam email corpus used within the project were both numerical as well as alphabetical. Approximately 50,000 emails were tested with the proposed models. The numerical corpuses (PU), had restrictions in terms of feature extraction as the words were replaced by numbers. But the alphabetical corpuses performed better in terms of extraction of the features and predicting the outcome.

Initially, WEKA [51] acted as a black box that ran the datasets on 14 different classification algorithms and provided the top 4 algorithms: Multinomial Naïve Bayes, Support Vector Machine, Random Forest and Decision Tree. These algorithms were then tested and experimented with Scikit-learns library and its modules. This resulted in upgrading the SVM module with SGD classifier, which acts the same as SVM but performs better on the large datasets. SGD was implemented using Python and experimented with feature extraction and stop words removal along with converting the tokens for the algorithms to process.

Genetic Algorithm worked better overall for both text-based datasets and numerical-based datasets than PSO. The PSO worked well for Multinomial Naïve Bayes and Stochastic Gradient Descent, whereas GA worked well for Random Forest and Decision Tree. Naïve Bayes algorithm was proved to have been the best algorithm for spam detection. This was concluded by evaluating the results for both numerical and alphabetical based dataset. The highest accuracy provided was 100% with GA optimization on randomised data distribution for 80:20 train and test split set on SpamAssassin dataset. In terms of F1-Score, precision and recall, Genetic Algorithm had more impact than PSO on MNB, SGD, RF and DT.

REFERENCES

[1] W. Feng, J. Sun, L. Zhang, C. Cao, and Q. Yang, "A support vector machine based Naive Bayes algorithm for spam filtering," in *Proc. IEEE 35th Int. Perform. Comput. Commun. Conf. (IPCCC)*, Dec. 2016, pp. 1–8, doi: 10.1109/pccc.2016.7820655.

[2] E. G. Dada, J. S. Bassi, H. Chiroma, S. M. Abdulhamid, A. O. Adetunmbi, and O. E. Ajibuwa, "Machine learning for email spam filtering: Review, approaches and open research problems," *Heliyon*, vol. 5, no. 6, Jun. 2019, Art. no. e01802, doi: 10.1016/j.heliyon.2019.e01802.

[3] W. Awad and S. ELseuofi, "Machine learning methods for spam E-Mail classification," *Int. J. Comput. Sci. Inf. Technol.*, vol. 3, no. 1, pp. 173–184, Feb. 2011, doi: 10.5121/ijcsit.2011.3112.

[4] S. Mohammed, O. Mohammed, and J. Fiaidhi, "Classifying unsolicited bulk email (UBE) using Python machine learning techniques," *Int. J. Hybrid Inf. Technol.*, vol. 6, no. 1, pp. 43–55, 2013. [Online]. Available: https://www.researchgate.net/publication/236970412_Classifying_Unsolicited_Bulk_Email_UBE_using_Python_Machine_Learning_Techniques

[5] A. Wijaya and A. Bisri, "Hybrid decision tree and logistic regression classifier for email spam detection," in *Proc. 8th Int. Conf. Inf. Technol. Electr. Eng. (ICITEE)*, Oct. 2016, pp. 1–4, doi: 10.1109/ICITEED.2016.7863267.

[6] K. Agarwal and T. Kumar, "Email spam detection using integrated approach of Naïve Bayes and particle swarm optimization," in *Proc. 2nd Int. Conf. Intell. Comput. Control Syst. (ICICCS)*, Jun. 2018, pp. 685–690, doi: 10.1109/ICCONS.2018.8662957.

[7] R. Belkebir and A. Guessoum, "A hybrid BSO-Chi2-SVM approach to arabic text categorization," in *Proc. ACS Int. Conf. Comput. Syst. Appl. (AICCSA)*, Ifran, Morocco, May 2013, pp. 1–7, doi: 10.1109/AICCSA.2013.6616437.

[8] A. I. Taloba and S. S. I. Ismail, "An intelligent hybrid technique of decision tree and genetic algorithm for E-Mail spam detection," in *Proc. 9th Int. Conf. Intell. Comput. Inf. Syst. (ICICIS)*, Cairo, Egypt, Dec. 2019, pp. 99–104, doi: 10.1109/ICICIS46948.2019.9014756.

[9] R. Karthika and P. Visalakshi, "A hybrid ACO based feature selection method for email spam classification," *WSEAS Trans. Comput*, vol. 14, pp. 171–177, 2015. [Online]. Available: https://www.wseas.org/multimedia/journals/computers/2015/a365705-553.pdf

[10] S. L. Marie-Sainte and N. Alalyani, "Firefly algorithm based feature selection for arabic text classification," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 32, no. 3, pp. 320–328, Mar. 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S131915781830106X

[11] E. A. Natarajan, S. Subramanian, and K. Premalatha, "An enhanced cuckoo search for optimization of bloom filter in spam filtering," *Global J. Comput. Sci. Technol.*, vol. 12, no. 1, pp. 75–81, 2012. Accessed: Jan. 18, 2020. [Online]. Available: https://globaljournals.org/GJCST_Volume12/12-An-Enhanced-Cuckoo-Search-for-Optimization.pdf

[12] A. Géron, *Hands-On Machine Learning With Scikit-Learn, Keras, and TensorFlow*, 2nd ed. Newton, MA, USA: O'Reilly Media, 2019, Ch. 1.

[13] (2019). *1. Supervised Learning—Scikit-Learn 0.22.2 Documentation*. Accessed: Oct. 9, 2019. [Online]. Available: https://scikit-learn.org/stable/supervised_learning.html

[14] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, and J. Vanderplas, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, Oct. 2011. [Online]. Available: http://www.jmlr.org/papers/volume12/pedregosa11a/pedregosa11a.pdf

[15] S. Zhu and F. Chollet. (2019). *Working With RNNs*. Accessed: Nov. 2, 2019. [Online]. Available: https://keras.io/guides/working_with_rnns/

[16] (2019). *TensorFlow Core | Machine Learning for Beginners and Experts*. Accessed: Nov. 2, 2019. [Online]. Available: https://www.tensorflow.org/overview

[17] (2019). *Spyder: The Scientific Python Development Environment—Documentation—Spyder 3 Documentation*. Accessed: Nov. 2, 2019. [Online]. Available: https://docs.spyder-ide.org/

[18] (2019). *User Guide—Anaconda Documentation*. Accessed: Nov. 9, 2019. [Online]. Available: https://docs.anaconda.com/ae-notebooks/user-guide/

[19] (2020). *Google Colaboratory*. Accessed: Mar. 18, 2020. [Online]. Available: https://colab.research.google.com

[20] S. Sawla. (2018). *Introduction to Naive Bayes for Classification*. Accessed: Oct. 9, 2019. [Online]. Available: https://medium.com/@srishtisawla/introduction-to-naive-bayes-for-classification-baefefb43a2d

[21] GeeksforGeeks. (2019). *Naive Bayes Classifiers-GeeksforGeeks*. Accessed: Nov. 10, 2019. [Online]. Available: https://www.geeksforgeeks.org/naive-Bayes-classifiers/

[22] G. Bonaccorso, *Machine Learning Algorithms*, 2nd ed. Birmingham, U.K.: Packt Publishing, 2018.

[23] G. Singh, B. Kumar, L. Gaur, and A. Tyagi, "Comparison between multinomial and Bernoulli Naïve Bayes for text classification," in *Proc. Int. Conf. Autom., Comput. Technol. Manage. (ICACTM)*, London, U.K., Apr. 2019, pp. 593–596, doi: 10.1109/ICACTM.2019.8776800.

[24] (2018). *Implementing 3 Naive Bayes classifiers in Scikit-Learn | Packt Hub*. Accessed: Nov. 13, 2019. [Online]. Available: https://hub.packtpub.com/implementing-3-naive-Bayes-classifiers-in-scikit-learn/

[25] (2019). *Sklearn.Linear_Model.SGDclassifier—Scikit-Learn 0.22.2 Documentation*. Accessed: Nov. 29, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html#sklearn.linear_model.SGDClassifier.

[26] A. Géron, *Hands-on Machine Learning With Scikit-Learn, Keras, and TensorFlow*, 2nd ed. Newton, MA, USA: O'Reilly Media, 2019, Ch. 6.

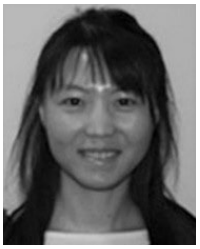
[27] T. Yiu. (2019). *Understanding Random Forest*. Medium. Accessed: Jan. 17, 2020. [Online]. Available: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

[28] (2020). *1.17. Neural Network Models (Supervised)—Scikit-Learn 0.22.2 Documentation*. Accessed: Mar. 17, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/neural_networks_supervised.html#neural-networks-supervised

[29] I. Androutsopoulos. (2000). *Ling-Spam*. Aueb.gr. Accessed: Oct. 11, 2019. [Online]. Available: http://www.aueb.gr/users/ion/data/lingspam_public.tar.gz
[30] I. Androutsopoulos, V. Metsis, and G. Paliouras. (2006). *The Enron-Spam Datasets*. Accessed: Oct. 11, 2019. [Online]. Available: http://www2.aueb.gr/users/ion/data/enron-spam/
[31] I. Androutsopoulos, G. Paliouras, and E. Michelakis. (2003). *PU Corpus*. Accessed: Oct. 1, 2019. [Online]. Available: http://www.aueb.gr/users/ion/data/PU123ACorpora.tar.gz
[32] I. Androutsopoulos, G. Paliouras, and V. Karkaletsis. (2003). *Learning to Filter Unsolicited Commercial E-Mail*. [Online]. Available: http://www2.aueb.gr/
[33] (2002). *Index of /Old/Publiccorpus*. Accessed: Oct. 11, 2019. [Online]. Available: https://spamassassin.apache.org/old/publiccorpus/
[34] N. Rusland, N. Wahid, S. Kasim, and H. Hafit, "Analysis of Naïve Bayes algorithm for email spam filtering across multiple datasets," in *Proc. IOP Conf. Ser., Mater. Sci. Eng.*, vol. 226, 2017, Art. no. 012091, doi: 10.1088/1757-899X/226/1/012091.
[35] (2019). *3.3. Metrics and Scoring: Quantifying the Quality Of Predictions—Scikit-Learn 0.22.2 Documentation*. Accessed: Dec. 31, 2019. [Online]. Available: https://scikit-learn.org/stable/modules/model_evaluation.html
[36] J. Lester. (2017). *Welcome to PySwarms's Documentation!—PySwarms 1.1.0 Documentation*. Accessed: Jan. 16, 2020. [Online]. Available: https://pyswarms.readthedocs.io/en/latest/index.html
[37] R. Olson. (2019). *Home—TPOT*. Accessed: Jan. 12, 2020. [Online]. Available: https://epistasislab.github.io/tpot/
[38] R. Shams and R. E. Mercer, "Classifying spam emails using text and readability features," in *Proc. IEEE 13th Int. Conf. Data Mining*, Dallas, TX, USA, Dec. 2013, pp. 657–666, doi: 10.1109/ICDM.2013.131.
[39] T. Kumareson. (2016). *Certain Investigations On Optimization Techniques to Enhance E-Mail Spam Classification*. Anna University. Accessed: Feb. 26, 2020. [Online]. Available: https://shodhganga.inflibnet.ac.in/handle/10603/181292
[40] H. Faris, I. Aljarah, and B. Al-Shboul, "A hybrid approach based on particle swarm optimization and random forests for e-mail spam filtering," in *Proc. Int. Conf. Comput. Collective Intell.*, 2016, pp. 498–508. [Online]. Available: https://www.researchgate.net/publication/304158714_A_Hybrid_Approach_based_on_Particle_Swarm_Optimization_and_Random_Forests_for_Email_Spam_Filtering
[41] F. Temitayo, O. Stephen, and A. Abimbola, "Hybrid GA-SVM for Efficient Feature Selection in E-mail Classification," *Comput. Eng. Intell. Syst.*, vol. 3, no. 3, pp. 17–28, 2012. [Online]. Available: https://www.researchgate.net/publication/257479733_Hybrid_GA-SVM_for_Efficient_Feature_Selection_in_E-mail_Classification
[42] A. Alghoul, S. Ajrami, and G. Jarousha, "Email classification using artificial neural network," *Int. J. Academic Eng. Res.*, vol. 2, no. 11, pp. 8–14, 2018. [Online]. Available: https://www.researchgate.net/publication/329307944_Email_Classification_Using_Artificial_Neural_Network
[43] M. Rathi and V. Pareek, "Spam mail detection through data mining–a comparative performance analysis," *Int. J. Mod. Edu. Comput. Sci.*, vol. 5, no. 12, pp. 31–39, Dec. 2013, doi: 10.5815/ijmecs.2013.12.05.
[44] S. R. Gomes, S. G. Saroar, M. Mosfaiul, A. Telot, B. N. Khan, A. Chakrabarty, and M. Mostakim, "A comparative approach to email classification using Naive Bayes classifier and hidden Markov model," in *Proc. 4th Int. Conf. Adv. Electr. Eng. (ICAEE)*, Dhaka, India, Sep. 2017, pp. 482–487, doi: 10.1109/ICAEE.2017.8255404.
[45] A. Yasin and A. Abuhasan, "An intelligent classification model for phishing email detection," *Int. J. Netw. Secur. Appl.*, vol. 8, no. 4, pp. 55–72, 2016. [Online]. Available: https://arxiv.org/abs/1608.02196
[46] A. Yüksel, S. Çankaya, and I. Üncü, "Design of a machine learning based predictive analytics system for spam problem," *Acta Phys. Polonica A*, vol. 132, no. 3, pp. 500–504, 2017. [Online]. Available: https://www.researchgate.net/profile/S_Fuat_Cankaya/publication/320320971_Design_of_a_Machine_Learning_Based_Predictive_Analytics_System_for_Spam_Problem/links/5a1c00e80f7e9be37f9c1ad1/Design-of-a-Machine-Learning-Based-Predictive-Analytics-System-for-Spam-Problem.pdf
[47] S. Sharma and A. Arora, "Adaptive approach for spam detection," *Int. J. Comput. Sci.*, vol. 10, no. 4, pp. 23–26, 2013. [Online]. Available: https://ijcsi.org/papers/IJCSI-10-4-1-23-26.pdf
[48] A. A. Akinyelu and A. O. Adewumi, "Classification of phishing email using random forest machine learning technique," *J. Appl. Math.*, vol. 2014, pp. 1–6, May 2014, doi: 10.1155/2014/425731.
[49] (2020). *6.2. Feature Extraction—Scikit-Learn 0.23.2 Documentation*. Accessed: Aug. 8, 2020. [Online]. Available: https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction
[50] (2019). *3.1. Cross-Validation: Evaluating Estimator Performance—Scikit-Learn 0.22 Documentation*. Accessed: Dec. 21, 2019. [Online]. Available: https://scikitlearn.org/stable/modules/cross_validation.html
[51] E. Frank, M. A. Hall, and I. H. Witten. *The WEKA Workbench Online Appendix for Data Mining: Practical Machine Learning Tools and Techniques*, 4th ed. San Mateo, CA, USA: Morgan Kaufmann, 2016.

SIMRAN GIBSON is currently pursuing the M.Comp. degree in computer networks and cyber security with Northumbria University, U.K. From 2018 to 2019, she was a year-long placement student at Nissan Manufacturing, U.K. She has been working with the Cybersecurity Department, her job entailed managing front line security operations such as IPS alerts, digital forensic investigation, malware triaging, and endpoint security alerts. Her recent awards include the First Prize in the Cyber Security Student Conference (CSSC 2020), the Ede and Ravenscroft Prize for best academic performance, and the Volunteering Role Recognition for student representative at Northumbria University.

BIJU ISSAC (Senior Member, IEEE) received the B.E. degree in electronics and communications engineering, the Master of Computer Applications (M.C.A.) degree, and the Ph.D. degree in networking and mobile communications. He has been an Academic Staff with Northumbria University, U.K., since 2018. He is research active and has authored more than 100 refereed conference papers, journal articles, and book chapters. His research interests include networks, cybersecurity, machine learning (text mining/image processing), and technology in education. He is on the Program Committee of many peer-reviewed international conferences and editorial boards of various journals.



LI ZHANG (Senior Member, IEEE) received the Ph.D. degree from the University of Birmingham, U.K. She is currently an Associate Professor and a Reader in computer science with Northumbria University, U.K. She has been serving as an Honorary Research Fellow with the University of Birmingham. Her research interests include artificial intelligence, machine learning, evolutionary computation, and deep learning. She has served as an Associate Editor for *Decision Support Systems*.

SEIBU MARY JACOB (Member, IEEE) received the B.Sc. and M.Sc. degrees in mathematics, the Post Graduate Diploma in Computer Applications (PGDCA) degree, the bachelor's degree in mathematics education (B.Ed.), and the Ph.D. degree in mathematics education. She is currently an Academic Staff teaching mathematics at Teesside University, U.K. She has authored more than 20 research publications as book chapters, journal articles, and conference papers. She is also a member of IET, IAENG, and IACSIT, for many years.