Spam Detection in Email Using Machine Learning

Prachika Agarwal
Pranjal Jain
New York University
NY,USA
pa2191@nyu.edu , pj2069@nyu.edu

***Abstract*—With over billions of email transactions every day, with over 30 percentage of them being spam, we are faced with a very crucial problem of classifying and protecting users from unwanted emails and targeted ads that encumbers the user from day-to-day tasks. Emails are checked by users almost instantly and hence become a prime target for many businesses that use them to send out unwanted emails. This problem is only worsened day by day as the spam sent out by businesses increases. It is the responsibility of the email hosting platforms to remove unwanted emails accurately. The platform needs to do this sensitively with high accuracy as they do not want to miss classify the email as spam, which was supposed to be important. The object of this project is to understand the challenge of the classification of spam documents and how different machine learning problems approach the task. We have implemented different algorithms and reviewed them using metrics to properly evaluate them.**

Fig. 1. Quad chart

Introduction

Email Spams are one of the most common issues we are facing in today's world. With the growing use of the internet, emails have become the most accessible forms of advertisements, thus giving way to sending emails to multiple recipients at the same time for commercial purposes. These emails which the user does not wish to receive are called spams. Receiving large amounts of spam emails can waste a lot of time for the user. Also, sometimes spam emails can contain malicious links which can cause phishing attacks and malware attacks intended on stealing confidential information from the user. Therefore, detecting such emails and reporting them becomes crucial for the safety of user's information as it's a major security concern.

In this project, we are using three classification algorithms which detect if the email is spam or not. The algorithms used are - Multinomial Naive Bayes, Logistic Regression and Random Forest Classifier. Here, we have taken the spam detection dataset from Kaggle, split it to train the model and test it. Later, we compare the accuracy of prediction of each of these models to decide which is the best classifier.

In naïve Bayes, we have approached it by understanding which bags of words in tokenized form can cause the probability to shift towards an email to be classified as spam once we fit on it. Some words have a higher probability of being used in a spam email and based on this fact we try to model a probability-based machine learning algorithm that understands which words have a higher chance and predict if those words exist. While a straightforward method, it can cause several misclassifications, that is why we use the Gaussian naïve Bayes algorithm to form from a normal distribution.

Random Forest or random decision tree is a supervised machine learning algorithm that works by having multiple decision trees trained differently to classify a single input and then polling it to find the average or mean outcome in-order to classify whether the input is spam or ham. We have further analyzed that we can use n-estimators or the number of trees used for polling and found the optimal value for it.

Logistic Regression works by categorizing whether an input is of a particular class. We have used binary logistic regression to find where we classify if it is spam or not. This is done by understanding what causes the likelihood of input to be spam. Since it is more direct as compared to a black box type classification (like Neural network) we get a direct understand of what causes the input to be spam.

E_Mail Spam Detection Model

Training the Model

The data frame is uploaded using pandas library pd.read_csv() command

	Unnamed	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n...	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see...	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar...	0
3	4685	spam	Subject: photoshop , windows , office . cheap ...	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t...	0
5	2949	ham	Subject: ehronline web address change\r\nthis ...	0
6	2793	ham	Subject: spring savings certificate - take 30 ...	0
7	4185	spam	Subject: looking for medication ? we ` re the ...	1
8	2641	ham	Subject: noms / actual flow for 2 / 26\r\nwe a...	0

Fig. 2. Picture of data set after importing

Data Cleaning and Pre_Processing

To clean the data the following steps were taken:

1) We converted the column 'Text' to type string to avoid disparities later on
2) We dropped the Unnamed column as it was not being used in our training model
3) Removed any duplicate values from the data using df.drop(inplace = True)
4) Dropped any null values we had in our data set by using df.dropna()
5) Imported the stopwords library for english language and removed any stopwords present in our dataset column 'Text'
6) Removed any punctuations present in the column 'Text' for a cleaner dataset

```
(7793, 3)
label         0
text          0
label_num     0
dtype: int64
```

	label	text	label_num
0	ham	Subject enron methanol meter 988291 follow ...	0
1	ham	Subject hpl nom january 9 2001 see attached ...	0
2	ham	Subject neon retreat ho ho ho around wonderf...	0
3	spam	Subject photoshop windows office cheap mai...	1
4	ham	Subject indian springs deal book teco pvr rev...	0
5	ham	Subject ehronline web address change message i...	0
6	ham	Subject spring savings certificate take 30 s...	0
7	spam	Subject looking medication best source diff...	1
8	ham	Subject noms actual flow 2 26 agree ...	0

Fig. 3. Data after cleaning and Pre_Processing

Splitting the dataset into training and test values

1) Sklearn library test_train_split was used to split the dataset into training and test with a test_size as 0.20 and randomstate as 11
2) We flattened X_train, X_test and y_train using values.ravel()

Using CountVectorization

1) Created a variable vectorizer calling CountVectorizer()
2) Defined a new variable named x_true_tr where we fit and transform X_train after flattening it, this is later converted into an array
3) Defined a new variable named x_true_ts where we fit X_test after flattening it, this is later converted into an array

Creating a model using Multinomial Naive Bayes

1) Create a variable named classifier_NB where we call GaussianNB()
2) Fit X_true_tr, y_train in classifer_NB
3) Predict the values of y_test using our model and store it in y_pred
4) Print the confusion matrix, accuracy score, classification report and mean squared error by passing y_test and y_pred as shown below

```
Confusion matrix
[[0.97643362 0.1048951 ]
 [0.03142184 0.86013986]]
Accuracy score
0.9550994227068633
Classification Report
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      1273
           1       0.89      0.86      0.88       286

    accuracy                           0.96      1559
   macro avg       0.93      0.92      0.92      1559
weighted avg       0.95      0.96      0.95      1559

Mean squared error
0.04490057729313662
```

Fig. 4. Metrics of Multinomial Guassian Naive Bayes Algorithm

Creating a model using Logistic Regression

1) Declare an empty array acc[ ], we will use this to store the accuracy value of the model with different C values
2) Create a loop for starting 1 and ranging till 5
   a) Create a variable named logreg where we call LogisticRegression(solver = 'liblinear', C = i)
   b) Fit X_true_tr, y_train in logreg
   c) Predict the values of y_test using our model and store it in y_pred
   d) Calculate the accuracy score of y_pred and y_test
   e) Add this score to the variable we previously created acc[ ]
   f) Wrote an if statement to store the y_pred value for the datapoint with maximum accuracy in y_pred_act
3) Plot the graph for accuracy vs diff C values
4) Print the confusion matrix, accuracy score, classification report and mean squared error by passing y_test and y_pred as shown below

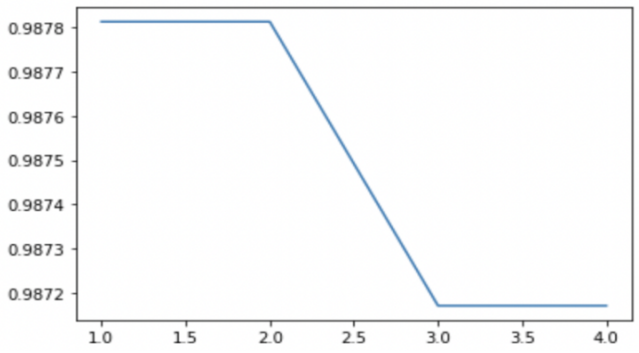


Fig. 5. Graph showing C vs Accuracy For Logistic Regression

```
Confusion Matrix is
[[0.99214454 0.03496503]
 [0.00706991 0.96853147]]
Accuracy is
0.9878127004490058
Classification Matrix is
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1273
           1       0.97      0.97      0.97       286

    accuracy                           0.99      1559
   macro avg       0.98      0.98      0.98      1559
weighted avg       0.99      0.99      0.99      1559

Mean squared error
0.012187299550994226
```

Fig. 6. Metrics of Logistic Regression

Creating a model using Random Forest Classifier

1) Declare an empty array acc[ ], we will use this to store the accuracy value of the model with different n_estimator values
2) Declare a variable iterator with range from 25:50, this iterator will be passed for different n_estimator values in the loop
3) Create a loop for variable i in iterator
    a) Create a variable named classifier_rf where we call RandomForestClassifier(n_estimators = i)
    b) Fit X_true_tr, y_train in classifier_rf
    c) Predict the values of y_test using our model and store it in y_pred
    d) Calculate the accuracy score of y_pred and y_test
    e) Add this score to the variable we previously created acc[ ]
    f) Wrote an if statement to store the y_pred value for the datapoint with maximum accuracy in y_pred_act
4) Plot the graph for accuracy vs diff n_estimator values
5) Print the confusion matrix, accuracy score, classification report and mean squared error by passing y_test and y_pred as shown below

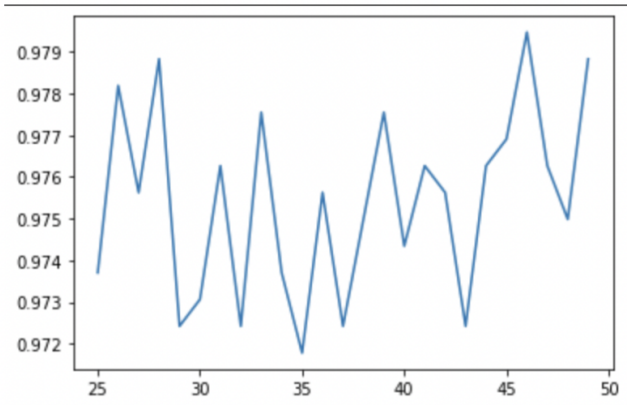


Fig. 7. Graph showing n_estimator vs Accuracy For Random Forest Classifier

```
Confusion Matrix is
[[0.99214454 0.03496503]
 [0.01728201 0.92307692]]
Accuracy is
0.9794740218088518
Classification Matrix is
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1273
           1       0.96      0.92      0.94       286

    accuracy                           0.98      1559
   macro avg       0.97      0.96      0.96      1559
weighted avg       0.98      0.98      0.98      1559

Mean Squared Error is
0.021167415009621552
```

Fig. 8. Metrics of Random Forest Classifier

ACCURACY COMPARISON OF OUR MODEL WITH EXISTING MODELS

Existing work

A Survey of Existing E-Mail Spam Filtering Methods Considering Machine Learning Techniques[1]

Author	Algorithms	Corpus or Datasets	Accuracy/ Performance
Subramaniam et al.	Naive Bayesian	Collection of spam emails from Google's Gmail Account	96.00% Accuracy Achieved
DeBarr et al.	Random Forest Algorithm	Custom Collection	95.2% Accuracy

Fig. 9. Summary of different existing email spam classification approaches regarding Machine Learning Techniques

Email Spam Detection Using Integrated Approach of Naïve Bayes and Particle Swarm Optimization[2]

Evaluating the Effectiveness of Machine Learning Methods for Spam Detection[3]

Evaluation Measures	Naïve Bayes
Precision (%)	88.71 %
Recall (%)	86.50 %
F-Measure (%)	87.59 %
Accuracy (%)	87.75 %

Fig. 10. Summary of Spam Detection with Naive Bayes mentioned in the paper

Algorithm	Accuracy	Precision	Recall	F-measure
Random forest	0,84	1	0,28	0,42
Logistic regression	0,99	0,98	0,96	0,97

Fig. 11. Summary of spam detection with Logistic Regression and Random Forest Algorithm mentioned in this paper

Our Work

Model Used	Multinomial Naive Bayes	Logistic Regression	Random Forest Classifier
Accuracy Score	95.50%	98.78%	97.94%

Fig. 12. Accuracy comparison of Multinomial Naive Bayes, Logistic Regression and Random Forest Classifier

Our project has been enhanced as we work with multiple datasets that have augmented our spam classification. It has helped us cover a variety of domains and topics that can be more encompassing of real-world scenarios, where spam could be of any form and topic. We applied additional pre-processing methodologies like removing useless words also known as stopwords, which contribute nothing to the message but encumber the model classifier. Furthermore, we have converted our dataset into tokenized vector forms that are more manageable to train on. We convert the words into a dictionary with an index associated with that word and train on the index values. We analyzed multiple classification models like Naïve Bayes, Logistic Regression, and Random Forest. This has helped us in getting a foundational understanding of how spam works and how different algorithms approach the task.

The reason for a higher accuracy model is the increased dataset and more parameter tuning. With Naive Bayes the higher probabilistic accuracy is due to a large variety of sources contributing to the domains expansion and search space. This helps in covering more topics and understanding more words that cause the emails to be labeled as spam. Logistic regression also takes advantage as it trains more. We have reduced overfitting by changing parameters like the C inverse regulariser and checking on multiple values. We also found that lib-learner was the optimal solver for the problem and gives the highest validation accuracy. The random forest proved to be giving higher accuracy as we have found the optimal number of trees to be 27, as we get high accuracy and low overfitting in our observations.

Conclusions

In this project, we have analyzed the three major algorithms for spam classification: Random-forest, Logistic Regression, and Naïve Bayes. Each has its strengths but we observe that Logistic regression has proved to be more accurate, albeit longer to train. We also noticed that increasing the dataset sources has proved to be covering more domains and topics from the emails and hence shows a higher validation accuracy. This is especially noticed in naïve Bayes, which can increase in accuracy significantly after getting data from multiple sources. In future works, we plan on covering more algorithms. Spam detection is a widely covered topic in Machine learning and requires careful understanding of how each algorithm works. We can further work on more algorithms such as Neural Networks, LSTMs, and Convolutional Neural networks but require special training equipment, especially when working on very large datasets.

Programming Material Referenced

1) Data Source - Data points from both of these datasets were taken to get one dataset with 7,793 samples
   a) https://www.kaggle.com/code/syamkakarla/spam-mail-classifier/data
   b) https://github.com/omaarelsherif/Email-Spam-Detection-Using-Machine-Learning/tree/main/Dataset
2) Training the model and cleaning up the data - https://www.kaggle.com/code/syamkakarla/spam-mail-classifier
   This kaggle submission was used as reference to see how to clean the data. In case of email spam detection we had to remove stop words from our data which was referenced from the kaggle submission. Along with that in this project we removed punctuations as well.
3) Scikit ibraries used as reference

References

[1] H. Bhuiyan, A. Ashiquzzaman, T. Juthi, S. Biswas, and J. Ara, "A survey of existing e-mail spam filtering methods considering machine learning techniques," 01 2018.

[2] K. Agarwal and T. Kumar, "Email spam detection using integrated approach of naïve bayes and particle swarm optimization," in *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*, 2018, pp. 685–690.

[3] Y. Kontsewaya, E. Antonov, and A. Artamonov, "Evaluating the effectiveness of machine learning methods for spam detection," EasyChair Preprint no. 6074, EasyChair, 2021.

Email_Spam_Detection_Model

May 17, 2022

Machine Learning - SPAM DETECTION - Prachika Agarwal(pa2191) and Pranjal Jain(pj2069)

[ ]:
```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import warnings
from google.colab import drive
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
import matplotlib.pyplot as plt

#Models
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier

warnings.filterwarnings('ignore')
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[ ]:
```
True
```

[ ]:
```
# Importing dataset
# Data taken from these two sources and combined together and filtered to␣
 ↪include 7793 samples:
# https://www.kaggle.com/code/syamkakarla/spam-mail-classifier/data
# https://github.com/omaarelsherif/Email-Spam-Detection-Using-Machine-Learning/
 ↪tree/main/Dataset
```

```
# drive link for dataset - https://drive.google.com/file/d/
 ↪1OuUCis-IZFpjJ3AvCip2WcGBQ61k_xrS/view?usp=sharing
drive.mount('/content/drive')
data_dir = '/content/drive/MyDrive/emailspamdetection/'

# Reading the csv file and printing top 10 values
df = pd.read_csv(data_dir+'spam_emails.csv')
df.head(10)
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:

	Unnamed	label	text	label_num
0	605	ham	Subject: enron methanol ; meter # : 988291\r\n…	0
1	2349	ham	Subject: hpl nom for january 9 , 2001\r\n(see…	0
2	3624	ham	Subject: neon retreat\r\nho ho ho , we ' re ar…	0
3	4685	spam	Subject: photoshop , windows , office . cheap …	1
4	2030	ham	Subject: re : indian springs\r\nthis deal is t…	0
5	2949	ham	Subject: ehronline web address change\r\nthis …	0
6	2793	ham	Subject: spring savings certificate - take 30 …	0
7	4185	spam	Subject: looking for medication ? we ` re the …	1
8	2641	ham	Subject: noms / actual flow for 2 / 26\r\nwe a…	0
9	1870	ham	Subject: nominations for oct . 21 - 23 , 2000\…	0

[ ]:
```
# Data Cleaning
# Reference for data cleaning - https://www.kaggle.com/code/syamkakarla/
 ↪spam-mail-classifier

# Converting
df = df.astype({'text' : str})
df.drop('Unnamed', axis = 1, inplace = True)

# Remove Duplicates
df.drop_duplicates(inplace = True)
print(df.shape)

# Check and remove null values
df = df.dropna()
print(df.isnull().sum())

# Remove stopwords and punctuations, print top 10 records
stop_word = stopwords.words('english')
df['text'] = df['text'].apply(lambda x: ' '.join ([word for word in x.split()␣
 ↪if word not in (stop_word)]))
df['text'] = df['text'].str.replace('[^\w\s]','')
df.head(10)
```

```
(7793, 3)
label        0
text         0
label_num    0
dtype: int64
```

[ ]:
	label	text	label_num
0	ham	Subject enron methanol meter 988291 follow …	0
1	ham	Subject hpl nom january 9 2001 see attached …	0
2	ham	Subject neon retreat ho ho ho around wonderf…	0
3	spam	Subject photoshop windows office cheap mai…	1
4	ham	Subject indian springs deal book teco pvr rev…	0
5	ham	Subject ehronline web address change message i…	0
6	ham	Subject spring savings certificate take 30 s…	0
7	spam	Subject looking medication best source diff…	1
8	ham	Subject noms actual flow 2 26 agree …	0
9	ham	Subject nominations oct 21 23 2000 see att…	0

[ ]:
```
# Define X and y values
X = df['text']
y = df['label_num']

#Getting the max no of characters in an email
max_value_column = df['text'].str.len().max()
print(max_value_column)
```

```
26305
```

[ ]:
```
# Splitting and reshaping training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 11)

# Flattening input data
X_train = X_train.values.ravel()
X_test = X_test.values.ravel()
y_train = y_train.values.ravel()
```

[ ]:
```
# Library used for Refernced - https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.feature_extraction.text.CountVectorizer.html

# Using CountVertorizer to convert train samples
data_tr = X_train
vectorizer = CountVectorizer()
X_true_tr = vectorizer.fit_transform(data_tr.ravel()).toarray()
print(X_true_tr.sum())

# Using CountVertorizer to convert test samples
data_ts = X_test
```

```
X_true_ts = vectorizer.transform(data_ts).toarray()
print(X_true_ts.sum())
```

```
848374
201080
```

[ ]:
```
# Guassian Multinomial Naive Bayes Classifier
# Library used for Refernced - https://scikit-learn.org/stable/modules/
  ↪naive_bayes.html

# Calling the classifier
classifier_NB = GaussianNB()

# Fitting the data
classifier_NB.fit(X_true_tr, y_train)

# Predicting the y values from X_true_ts
y_pred = classifier_NB.predict(X_true_ts)
```

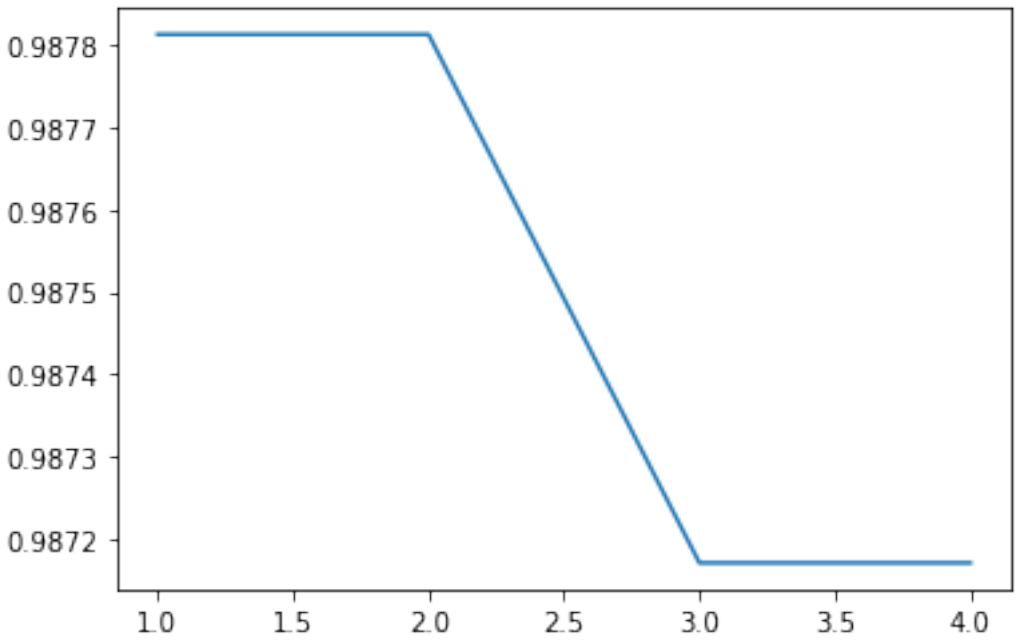
[ ]:
```
# Print metrics for Naive Bayes

# Confusion matrix
cm_NB = confusion_matrix(y_test, y_pred)
cm_NB = cm_NB/cm_NB.astype(float).sum(axis = 1)
print('Confusion matrix')
print(cm_NB)

# Accuracy score
acc_NB = accuracy_score(y_test, y_pred)
print('Accuracy score')
print(acc_NB)

# Classification Report
classreport_NB = classification_report(y_test,y_pred)
print('Classification Report')
print(classreport_NB)

# Mean Squared Error
mse_NB = np.square(np.subtract(y_test,y_pred)).mean()
print('Mean squared error')
print(mse_NB)
```

```
Confusion matrix
[[0.97643362 0.1048951 ]
 [0.03142184 0.86013986]]
Accuracy score
0.9550994227068633
Classification Report
```

```
              precision    recall  f1-score   support

           0       0.97      0.98      0.97      1273
           1       0.89      0.86      0.88       286

    accuracy                           0.96      1559
   macro avg       0.93      0.92      0.92      1559
weighted avg       0.95      0.96      0.95      1559

Mean squared error
0.04490057729313662
```

[ ]:
```
# Logistic Regression
# Library used for Refernced - https://scikit-learn.org/stable/modules/
  ↪generated/sklearn.linear_model.LogisticRegression.html

acc_log = []
y_pred_act = []
for i in range(1,5):
  # Calling the classifier
  logreg = LogisticRegression(solver = 'liblinear' , C = i)
  # Fitting the data
  logreg.fit(X_true_tr, y_train)
  # Predicting the y values from X_true_ts
  y_pred = logreg.predict(X_true_ts)
  # Calculating accuracy score for each C value
  acc = accuracy_score(y_test, y_pred)
  # Adding the accuracy score to a list
  acc_log.append(acc)
  # Storing the y_pred value for C with maximum accuracy
  if acc == max(acc_log):
    y_pred_act = y_pred

# Plotting C vs accuracy graph
plt.plot([1,2,3,4], acc_log)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f8fe837c350>]
```



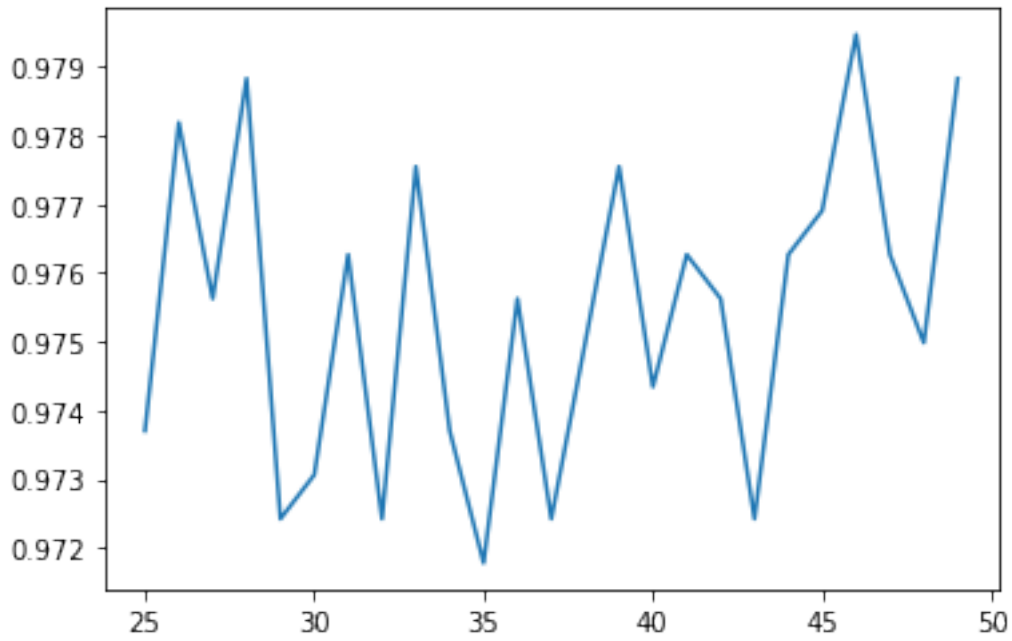
[ ]:
```
# Metrics of Logistic Regression

# Confusion matrix
cm_lr = confusion_matrix(y_test, y_pred_act)
cm_lr = cm_lr/cm_lr.astype(float).sum(axis = 1)
print('Confusion Matrix is')
print(cm_lr)

# Accuracy score
acc_lr = max(acc_log)
print('Accuracy is')
print(acc_lr)

# Classification Report
classreport_lr = classification_report(y_test, y_pred_act)
print('Classification Matrix is')
print(classreport_lr)

# Mean Squared Error
mse_lr = np.square(np.subtract(y_test, y_pred_act)).mean()
print('Mean squared error')
print(mse_lr)
```

```
Confusion Matrix is
[[0.99214454 0.03496503]
 [0.00706991 0.96853147]]
```

```
Accuracy is
0.9878127004490058
Classification Matrix is
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      1273
           1       0.97      0.97      0.97       286

    accuracy                           0.99      1559
   macro avg       0.98      0.98      0.98      1559
weighted avg       0.99      0.99      0.99      1559

Mean squared error
0.012187299550994226
```

[ ]:
```
# Random Forest
# Library used for Refernced - https://scikit-learn.org/stable/modules/
 ↪generated/sklearn.ensemble.RandomForestClassifier.html

acc_log = []
y_pred_act = []
iterator = list(range(25,50))
for i in iterator:
  # Calling the classifier
  classifier_rf = RandomForestClassifier(n_estimators = i)
  # Fitting the data
  classifier_rf.fit(X_true_tr, y_train)
  # Predicting the y values from X_true_ts
  y_pred = classifier_rf.predict(X_true_ts)
  # Calculating accuracy score for each C value
  acc_i = accuracy_score(y_test, y_pred)
  # Adding the accuracy score to a list
  acc_log.append(acc_i)
  # Storing the y_pred value for n_estimator with maximum accuracy
  if acc_i == max(acc_log):
    y_pred_act = y_pred

# Plotting iterator vs accuracy graph
plt.plot(iterator, acc_log)
```

[ ]:
```
[<matplotlib.lines.Line2D at 0x7f8fe5178250>]
```



[ ]:
```
# Random Forest Metrics

# Confusion matrix
cm_rf = confusion_matrix(y_test, y_pred_act)
cm_rf = cm_rf/cm_rf.astype(float).sum(axis = 1)
print('Confusion Matrix is')
print(cm_rf)

# Accuracy score
acc_rf = max(acc_log)
print('Accuracy is')
print(acc_rf)

# Classification Report
classreport_rf = classification_report(y_test, y_pred)
print('Classification Matrix is')
print(classreport_rf)

# Mean Squared Error
mse_rf = np.square(np.subtract(y_test, y_pred)).mean()
print('Mean Squared Error is')
print(mse_rf)
```

```
Confusion Matrix is
[[0.99214454 0.03496503]
 [0.01728201 0.92307692]]
```

```
Accuracy is
0.9794740218088518
Classification Matrix is
              precision    recall  f1-score   support

           0       0.98      0.99      0.99      1273
           1       0.96      0.92      0.94       286

    accuracy                           0.98      1559
   macro avg       0.97      0.96      0.96      1559
weighted avg       0.98      0.98      0.98      1559

Mean Squared Error is
0.021167415009621552
```