

# **Machine Learning**

## **Project Report (SMS Spam Classifier)**



**Name: Irfan Arshad**

**Reg# B21F0063AI019**

**Instructor: Dr Arshad Iqbal**

**Department of ITCS**

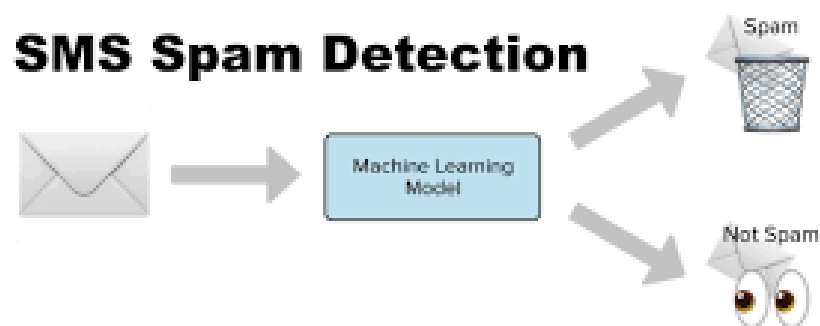
**PAK-AUSTRIA FACHHOCHSCHULE:  
INSTITUTE OF APPLIED SCIENCES AND  
TECHNOLOGY**

## ABSTRACT

Over recent years, as the popularity of mobile phone devices has increased, Short Message Service (SMS) has grown into a multi-billion-dollar industry. At the same time, reducing the cost of messaging services has resulted in growth in unsolicited commercial advertisements (spam) being sent to mobile phones. In parts of Asia, up to 30% of text messages were spam in 2012. Lack of real databases for SMS spam, short length of messages limited features, and their informal language are the factors that may cause the established email filtering algorithms to underperform in their classification.

In this project, a database of real SMS Spam from the UCI Machine Learning repository is used, and after pre-processing and feature extraction, different machine learning techniques are applied to the database. Finally, the results are compared and the best algorithm for spam filtering for text messaging is introduced. Algorithms used in this technique are: Logistic regression (LR), K-nearest neighbor (K-NN), Random Forest Classifier (RFC), Naïve Bayes (NB), and Decision tree (DT) used for the classification of spam messages in mobile device communication. The SMS spam collection set is used for testing the method.

## INTRODUCTION:



Short Message Services (SMS) is far more than just a technology for a chat. SMS technology evolved out of the global system for mobile communications standard, an internationally accepted. Spam is the abuse of electronic messaging systems to send

unsolicited messages in bulk indiscriminately. While the most widely recognized form of spam is email spam, the term is applied to similar abuses in other media and mediums. SMS Spam in the context is very similar to email spams, typically, unsolicited bulk messaging with some business interest.

SMS spam is used for commercial advertising and spreading phishing links. Commercial spammers use malware to send SMS spam because sending SMS spam is illegal in most countries. Sending spam from a compromised machine reduces the risk to the spammer because it obscures the provenance of the spam. SMS can have a limited number of characters, which includes alphabets, numbers, and a few symbols. A look through the messages shows a clear pattern. Almost all of the spam messages ask the users to call a number, reply by SMS, or visit some URL. This pattern is observable in the results obtained by a simple SQL query on the spam. The low price and the high bandwidth of the SMS network have attracted a large amount of SMS spam. 3

Every time SMS spam arrives at a user's inbox, the mobile phone alerts the user to the incoming message. When the user realizes that the message is unwanted, he or she will be disappointed, and also SMS spam takes up some of the mobile phone's storage.



SMS spam detection is an important task where spam SMS messages are identified and filtered. As more significant numbers of SMS messages are communicated every day, it is challenging for a user to remember and correlate the newer SMS messages received in context to previously received SMS. Thus, using the knowledge of artificial intelligence with the amalgamation of machine learning, and data mining we will try to develop web-based SMS text spam or ham detector.

A number of major differences exist between spam-filtering in text messages and emails. Unlike emails, which have a variety of large datasets available, real databases for SMS spams are very limited. Additionally, due to the small length of text messages, the number of features that can be used for their classification is far smaller than the corresponding number in emails. Here, no header exists as well. Additionally, text messages are full of abbreviations and have much less formal language than what one would expect from emails. All of these factors may result in serious degradation in performance of major email spam filtering algorithms applied to short text messages.

## **Problem Statement:**

Short Message Service (SMS) is a popular mobile application used for personal and business communication worldwide. It has become a billion-dollar industry for text marketing, customer interaction, business updates, and reminders [1]. Along with these applications of short messages, there are many text messaging mobile phone applications like WhatsApp, Telegram where people communicate daily, and nowadays many businesses are adopting these platforms to communicate with customers. Hence it is expected that this short message industry will grow enormously in the future. However, alongside legitimate SMSs, a significant number of spam messages flood in, serving various purposes, most of which are fraudulent. Filtering these spam SMS accurately is a challenging task that can benefit human lives both psychologically and financially.

## **Flow Chart:**

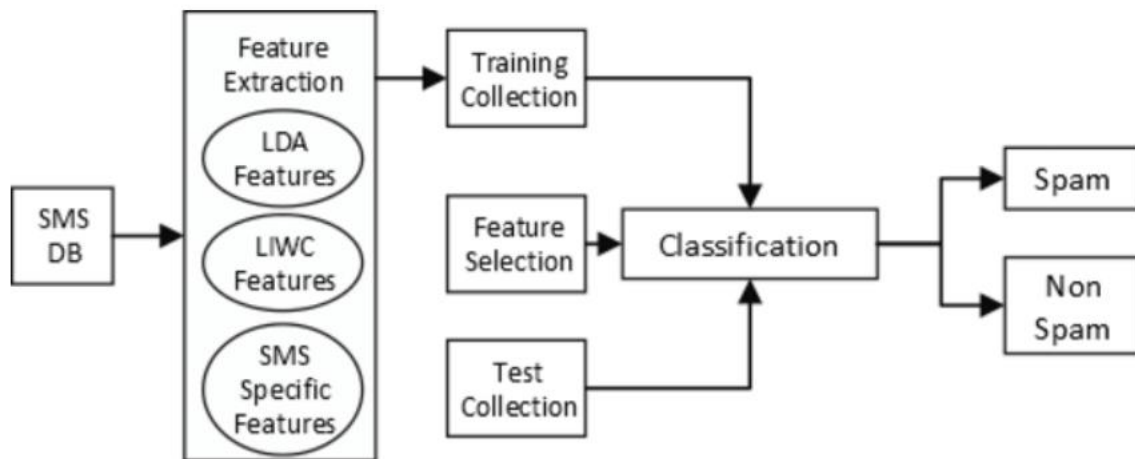


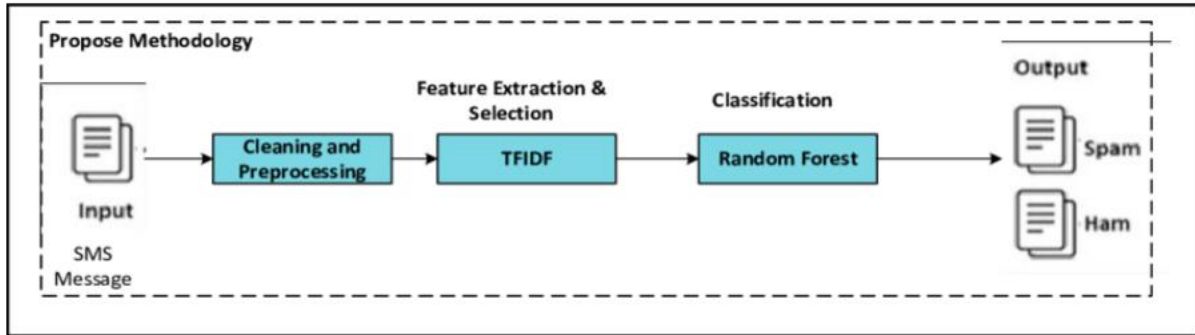
Fig-1.3 flow chart of sms spam detection

This is three parts of a bold series, where we will understand the in and out of spam or ham classifier from the aspect of Artificial Intelligence concepts, work with various classification algorithms in Jupyter Notebook, and select the one algorithm based on performance criteria. Then, we will develop the Python web-based SMS text spam or ham detector.

## REQUIREMENTS:

Anaconda Jupyter Notebook Anaconda3 Jupyter Notebook is used to write and execute the Python code using machine learning algorithms such as Naive Bayes Theorem, Logistic Regression, Decision Trees, K- Nearest Neighbors, and Random Forest.

## METHODOLOGY:



- **Dataset:**

The dataset is collected from the UCI [20] or [21] and this dataset has 2 columns, a label, and an SMS. The label column says that SMS is either “spam” or “ham” (i.e., not spam). The SMS column has the raw text of the SMS and each row in the dataset contains the raw text of one SMS and its associated label. This dataset contains 5574 messages/rows.

**Table 1. Detail Of The Dataset.**

Source	Total Messages
The Grumbletext Web site	425
NUS SMS Corpus (NSC)	3375
Caroline Tag's Ph.D. Thesis	450
Spam Corpus v.0.1 Big	1324
<b>Total</b>	<b>5574</b>

The main reason for choosing this dataset is combined by randomly sampling from different resources hence it gives exposure to different scenarios of ham and spam SMSs. In statistics, it is proven that random sampling is the best.

## INTRODUCTION TO THE MACHINE LEARNING ALGORITHMS:

- **K-Nearest Neighbors:**

k-nearest neighbor can be applied to classification problems as a simple instance-based learning algorithm. In this method, the label for a test sample is predicted based on the majority vote of its k nearest neighbors.

- **Random Forest:**

Random forests is an averaging ensemble method for classification. The ensemble is a combination of decision trees built from a bootstrap sample from training set. Additionally, in building the decision tree, the split which is chosen when splitting

a node is the best split only among a random set of features. This will increase the bias of a single model, but the averaging reduces the variance and can compensate for the increase in bias too. Consequently, a better model is built. In this work, the implementation of random forests in the scikitlearn python library is used, which averages the probabilistic predictions. Two number of estimators are simulated for this method. With 10 estimators, the overall error is 2.16%, SC is 87.7 %, and BH is 0.73%. Using 100 estimators will result in overall error of 1.41 %, SC of 92.2 %, and BH of 0.51 %. We observe that comparing to the naive Bayes algorithm, although the complexity of the model is increased, yet the performance does not show any improvement.

- **MultinomialNB:**

Multinomial Naive Bayes (MultinomialNB) is a machine learning algorithm used for text classification tasks. It is based on the Bayes theorem and assumes that the features (words) in a document are conditionally independent given the class label. MultinomialNB works by computing the probability of each class label given the occurrence of the features (words) in the input document. It is particularly useful for text classification tasks because it can handle sparse data and works well with large feature spaces. It is commonly used for tasks such as spam filtering, sentiment analysis, and topic classification.

- **Decision Tree:**

A decision tree is a machine-learning algorithm used for classification and regression. It works by recursively partitioning data into subsets based on a chosen feature and building a tree-like model of decisions until a stopping criterion is met. Each node represents a decision based on a feature and each branch represents a possible outcome. Decision trees are easy to interpret and handle categorical and numerical data. However, they can overfit and small changes in training data can lead to different structures. They are popular for tasks such as fraud detection, customer churn prediction, and medical diagnosis.

- **Logistic Regression:**

Logistic Regression is a machine learning algorithm used for binary classification. It models the probability of an input belonging to a certain class using a sigmoid

function applied to a linear combination of the input features. During training, it learns the coefficients that minimize the log-loss function. Logistic Regression is simple, efficient, and can handle both numerical and categorical input features. It is commonly used for tasks such as credit risk assessment, fraud detection, and medical diagnosis.

## **Data Preprocessing:**

Now that we have a basic understanding of what our dataset looks like, let's convert our labels to binary variables, 0 to represent 'ham'(i.e. not spam) and 1 to represent 'spam' for ease of computation. You might be wondering why do we need to do this step? The answer to this lies in how scikit-learn handles inputs. Scikit-learn only deals with numerical values and hence if we were to leave our label values as strings, scikit-learn would do the conversion internally (more specifically, the string labels will be cast to unknown float values).<sup>12</sup> Our model would still be able to make predictions if we left our labels as strings but we could have issues later when calculating performance metrics, for example when calculating our precision and recall scores. Hence, to avoid unexpected 'gotchas' later, it is good practice to have our categorical values be fed into our model as integers.

## **Bag of words:**

What we have here in our data set is a large collection of text data (5,572 rows of data). Most ML algorithms rely on numerical data to be fed into them as input, and email/sms messages are usually text-heavy. Here we'd like to introduce the Bag of Words (BOW) concept which is a term used to specify the problems that have a 'bag of words' or a collection of text data that needs to be worked with. The basic idea of BOW is to take a piece of text and count the frequency of the words in that text. It is important to note that the BOW concept treats each word individually and the order in which the words occur does not matter. Using a process which we will go through now, we can convert a collection of documents to a matrix, with each document being a row and each word(token) being the column, and the corresponding (row, column) values being the frequency of occurrence of each word or token in that document.



## Data preprocessing with CountVectorizer():

Some of the important parameters of CountVectorizer():

1. lowercase = True The lowercase parameter has a default value of True which converts all of our text to its lowercase form.
2. Token pattern = (?u)\b\w\w+\b The token pattern parameter has a default regular expression value of (?u)\b\w\w+\b which ignores all punctuation marks and treats them as delimiters, while accepting alphanumeric strings of length greater than or equal to 2, as individual tokens or words.
3. Stop words The stop words parameter, if set to English will remove all words from our document set that match a list of English stop words which is defined in scikit-learn. Considering the size of our dataset and the fact that we are dealing with SMS messages and not larger text sources like e-mail, we will not be setting this parameter value.

## Training and testing sets:

Now that we have understood how to deal with the Bag of Words problem we can get back to our dataset and proceed with our analysis. Our first step in this regard would be to split our dataset into a training and testing set so we can test our model later.

Instructions: Split the dataset into a training and testing set by using the train\_test\_split method in sklearn. Split the data using the following variables:

- X\_train is our training data for the 'sms\_message' column.
- Y\_train is our training data for the 'label' column
- X\_test is our testing data for the 'sms\_message' column.
- y\_test is our testing data for the 'label' column Print out the number of rows we have in each our training and testing data.

## Implementation of Machine Learning Algorithms:

I will use sklearn's sklearn.naive\_bayes method to make predictions on our dataset for SMS Spam Detection. Specifically, we will be using the multinomial Naive Bayes implementation. This particular classifier is suitable for classification with discrete features. It takes in integer word counts as its input.

## Evaluating our model:

- **Evaluation Metrics:**

In machine learning, a confusion matrix is a table used to evaluate the performance of a classification model. It is a matrix that summarizes the actual and predicted class labels for a set of test data. As shown in Figure 3, the confusion matrix shows the number of true positives, false positives, true negatives, and false negatives for each class. True positives (TP) are the number of correct predictions of the positive class, false positives (FP) are the number of incorrect predictions of the positive class, true negatives (TN) are the number of correct predictions of the negative class, and false negatives (FN) are the number of incorrect predictions of the negative class. A confusion matrix can be used to calculate various evaluation metrics such as accuracy, precision, recall, and F1 score.

Figure 4. Confusion Matrix Of Spam And Ham.

Confusion Matrix	Predicted HAM	Predicted SPAM
Actually HAM	TN (True Negative)	FP (False Positive)
Actually SPAM	FN (False Negative)	TP (True Positive)

To evaluate and compare different models, accuracy, precision, and recall are used as evaluation metrics in this research.

### **Accuracy:**

Accuracy measures the overall proportion of SMS messages that are correctly classified by the model, both spam and legitimate. While accuracy is an important metric, it is not as useful in SMS spam detection as it can be skewed by the high proportion of legitimate messages in the dataset. We can calculate accuracy using the formula below.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}$$

### **Precision:**

Precision measures the proportion of SMS messages classified as spam that are actually spam. In other words, it measures how many true positives (spam

messages correctly classified as spam) there are relative to false positives (legitimate messages incorrectly classified as spam). In SMS spam detection, precision is an important metric as it ensures that users are not receiving too many false positives (legitimate messages incorrectly classified as spam) that could result in the loss of important messages. The precision of the model can be calculated using TP, and FP values given in the confusion matrix.

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

**Recall:**

Recall, on the other hand, measures the proportion of actual spam messages that were correctly classified as spam. In other words, it measures how many true positives (spam messages correctly classified as spam) there are relative to false negatives (spam messages incorrectly classified as legitimate). The recall is also an important metric in SMS spam detection, as it ensures that the model is able to capture all spam messages and not miss any that could result in users receiving unwanted messages. The formula of recall is shown below.

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

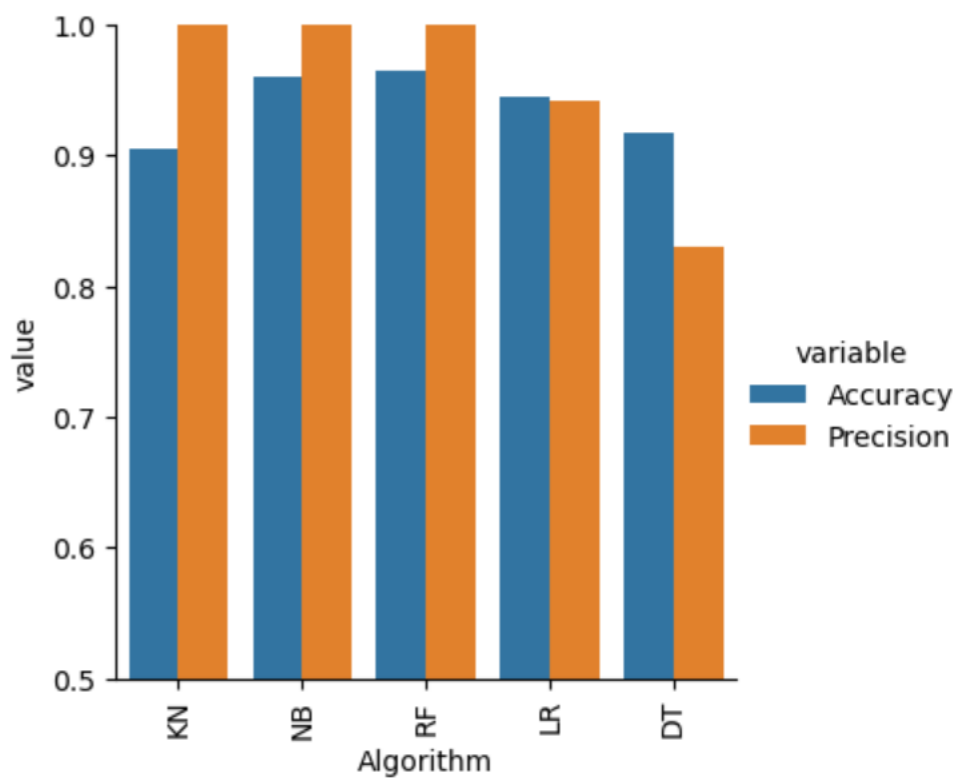
**Conclusion:**

One of the major advantages that Naive Bayes has over other classification algorithms is its ability to handle an extremely large number of features. In our case, each word is treated as a feature and there are thousands of different words. Also, it performs well even with the presence of irrelevant features and is relatively unaffected by them. The other major advantage it has is its relative simplicity. Naive Bayes' works well right out of the box and tuning its parameters is rarely ever necessary, except usually in cases where the distribution of the data is known. It rarely ever overfits the data. Another important advantage is that its model training and prediction times are very fast for data it can handle. All in all, Naive Bayes' is a gem of an algorithm!

**Compare Different ML and DL models:**

Below are different machine learning models compared.

	Algorithm	Accuracy	Precision
0	KN	0.905314	1.000000
1	NB	0.960386	1.000000
4	RF	0.965217	1.000000
3	LR	0.944928	0.941748
2	DT	0.916908	0.829787



# Performance Measure

$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{\text{Total Number of Test Data}}$$

$$\text{Spams caught (SC)} = \frac{\text{False negative cases}}{\text{Number of Spams}}$$

$$\text{Blocked hams (BH)} = \frac{\text{False Positive cases}}{\text{Number of Hams}}$$

## TESTING:

```
Instantiate the CountVectorizer method

In [45]:
count_vector = CountVectorizer()

Fit the training data and then return the matrix

In [46]:
training_data = count_vector.fit_transform(X_train)

In [47]: training_data
Out[47]: <4457x7774 sparse matrix of type '<class 'numpy.int64'>'
         with 59357 stored elements in Compressed Sparse Row format>

In [48]: # Transform testing data and return the matrix.
testing_data = count_vector.transform(X_test)

In [49]: testing_data
Out[49]: <1115x7774 sparse matrix of type '<class 'numpy.int64'>'
         with 13599 stored elements in Compressed Sparse Row format>
```

Fig.1 Testing the data.

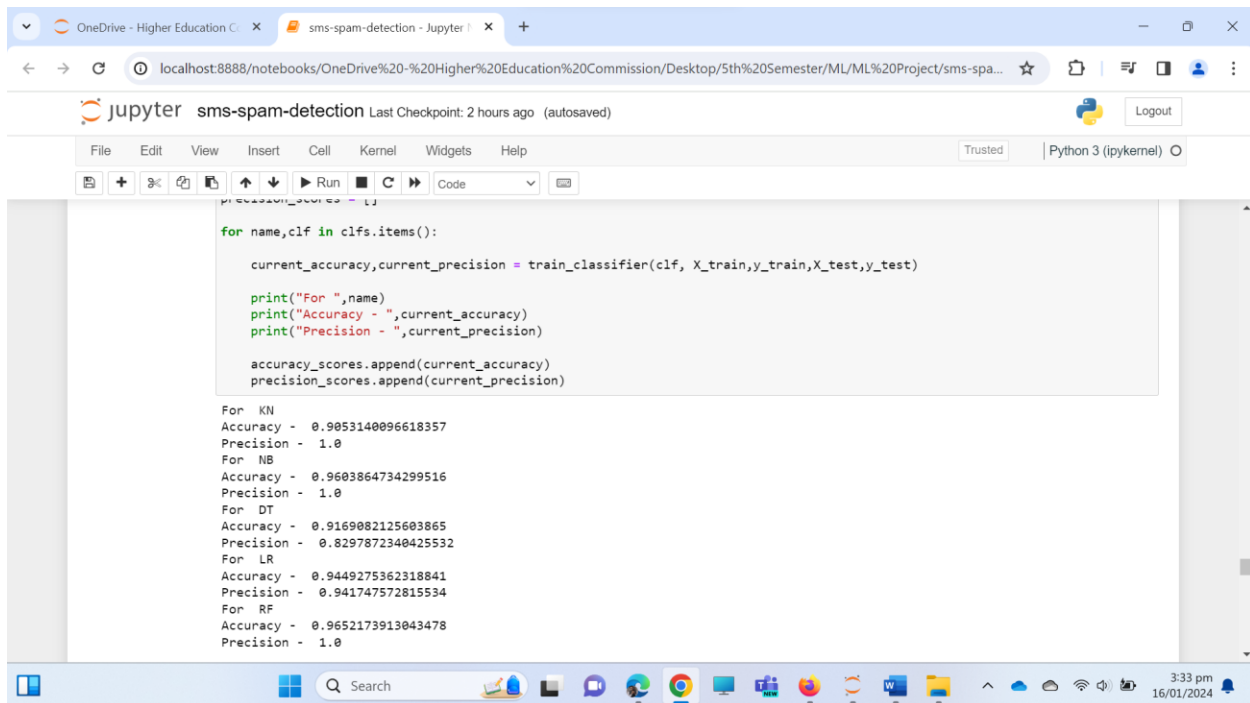
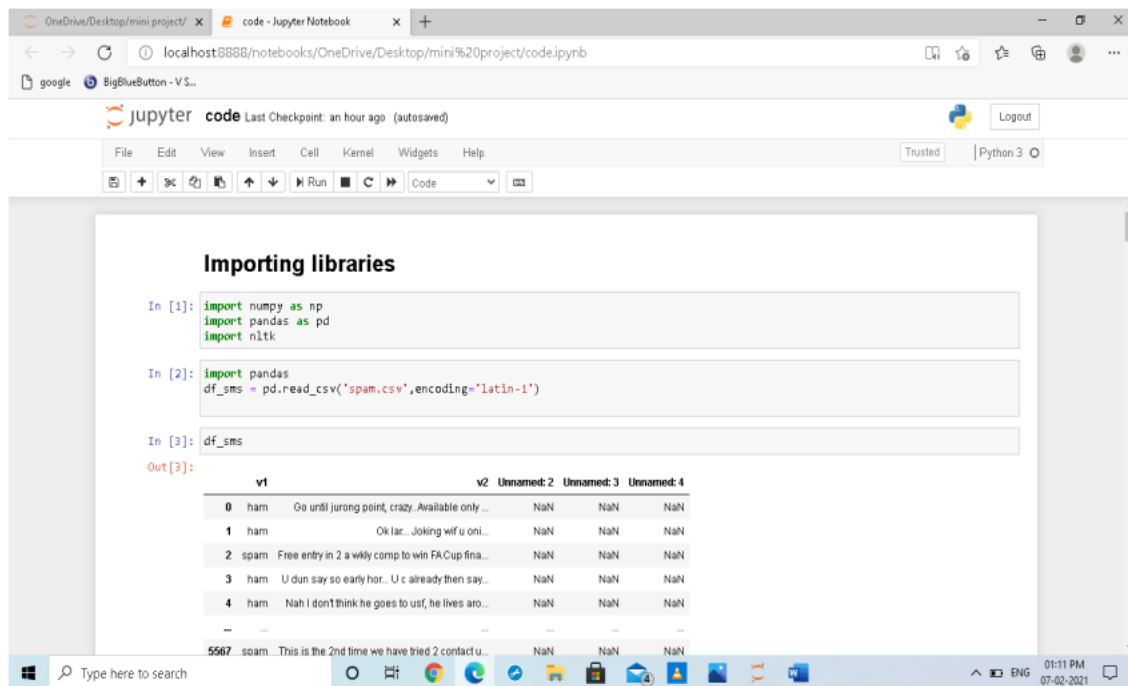


Fig.2 Evaluating the model.

## OUTPUT SCREENS:



Importing libraries

```
4 ham Nah I dont think he goes to usf, he lives aro... NaN NaN NaN
...
5567 spam This is the 2nd time we have tried 2 contact u... NaN NaN NaN
5568 ham Will l_b going to esplanade h home? NaN NaN NaN
5569 ham Pity * was in mood for that. So...any other s... NaN NaN NaN
5570 ham The guy did some bitching but i acted like i'd... NaN NaN NaN
5571 ham Rofl. Its true to its name NaN NaN NaN

5572 rows x 5 columns

In [4]: df_sms.head()
Out[4]:
```

	v1	v2	Unnamed: 2	Unnamed: 3	Unnamed: 4
0	ham	On until juring point, crazy. Available only ...	NaN	NaN	NaN
1	ham	Ok lat...Joking wif u onl...	NaN	NaN	NaN
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...	NaN	NaN	NaN
3	ham	U dun say so early hor... U c already then say...	NaN	NaN	NaN
4	ham	Nah I dont think he goes to usf, he lives aro...	NaN	NaN	NaN

```
In [5]: df_sms.head()
Out[5]:
```

## Viewing the Data

```
5568 ham Will l_b going to esplanade h home?
5569 ham Pity * was in mood for that. So...any other s...
5570 ham The guy did some bitching but i acted like i'd...
5571 ham Rofl. Its true to its name

5572 rows x 2 columns

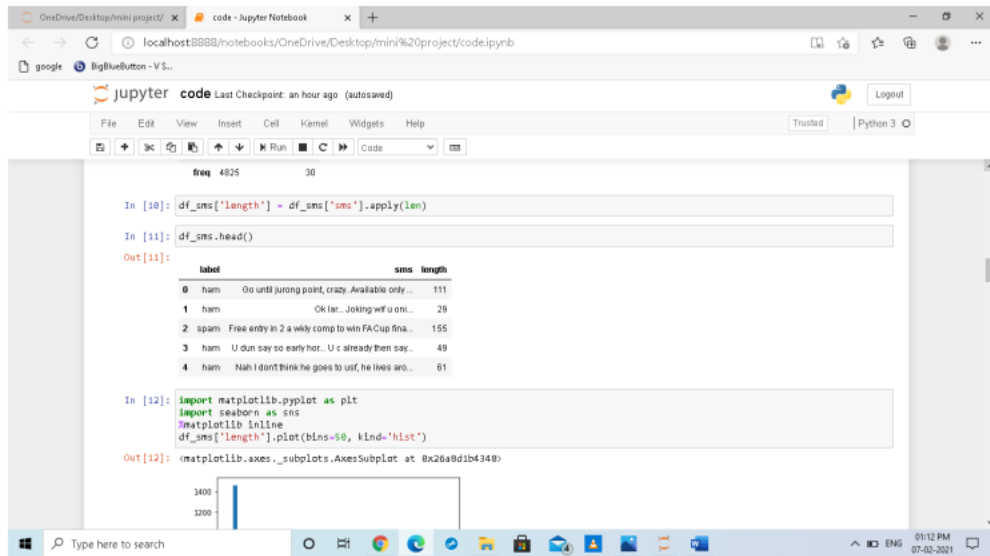
In [8]: print(len(df_sms))
5572

In [9]: df_sms.describe()
Out[9]:
```

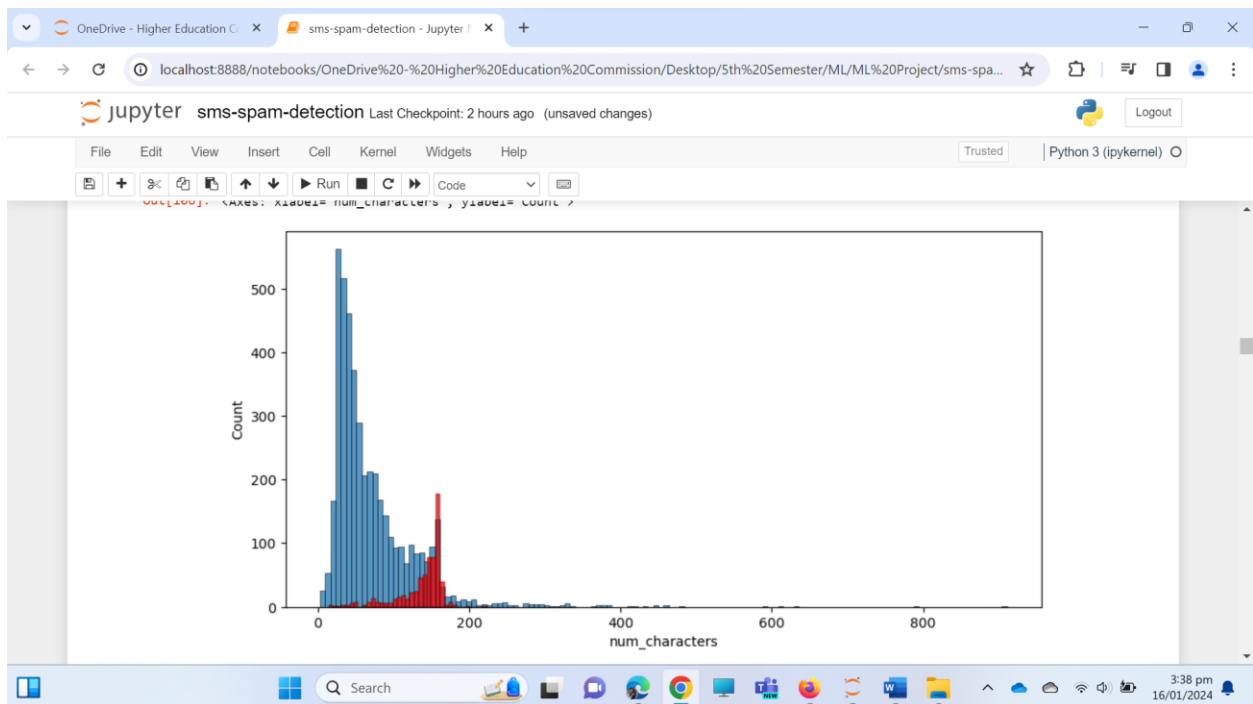
	label	sms
count	5572	5572
unique	2	5169
top	ham	Sorry,I'll call later
freq	4825	30

```
In [10]: df_sms['length'] = df_sms['sms'].apply(len)
In [11]: df_sms.head()
```

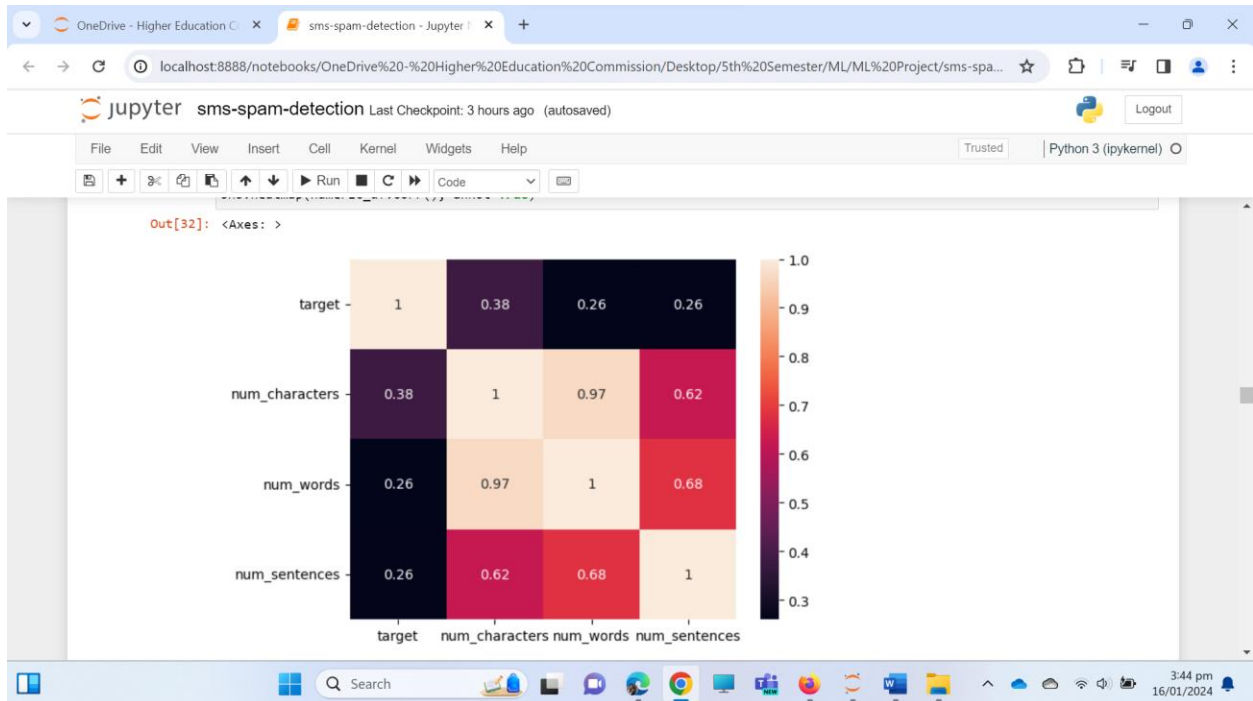
## Viewing the Unique Data



## Importing libraries to plot graphs







Correlation Matrix

```
df_sms.loc[:, 'label'] = df_sms.label.map({'ham': 0, 'spam': 1})
print(df_sms.shape)
df_sms.head()
```

Out[14]:

	label	sms	length
0	0	Go until jurong point, crazy.. Available only...	111
1	0	Ok lar... Joking wif u oni...	29
2	1	Free entry in 2 a wkly comp to win FA Cup fina...	155
3	0	U dun say so early hor... U c already then say...	49
4	0	Nah I don't think he goes to usf, he lives aro...	61

### Implementation of Bag of Words Approach

#### Step 1: Convert all strings to their lower case form.

```
documents = ['Hello, how are you!',
              'Win money, win from home.',
              'Call me now.',
              'Hello, Call hello you tomorrow?']
```

Converting all Strings to their lower Form

The screenshot shows a Jupyter Notebook interface with the following content:

### Implementation of Bag of Words Approach

#### Step 1: Convert all strings to their lower case form.

```
In [15]: documents = ['Hello, how are you!',  
                    'Win money, win from home.',  
                    'Call me now.',  
                    'Hello, Call hello you tomorrow?']
```

```
In [16]: lower_case_documents = []  
lower_case_documents = [d.lower() for d in documents]  
print(lower_case_documents)
```

```
['hello, how are you!', 'win money, win from home.', 'call me now.', 'hello, call hello you tomorrow?']
```

#### Step 2: Removing all punctuations

```
In [17]: sans_punctuation_documents = []  
import string
```

```
for i in lower_case_documents:  
    sans_punctuation_documents.append(i.translate(str.maketrans("", "", string.punctuation)))
```

The notebook is running on a local host at localhost8888. The Windows taskbar at the bottom shows the time as 01:12 PM on 07-02-2021.

Removing all punctuation

The screenshot shows the Jupyter Notebook interface with the following content:

#### Step 2: Removing all punctuations

```
In [17]: sans_punctuation_documents = []  
import string
```

```
for i in lower_case_documents:  
    sans_punctuation_documents.append(i.translate(str.maketrans("", "", string.punctuation)))
```

```
sans_punctuation_documents
```

```
Out[17]: ['hello how are you',  
          'win money win from home',  
          'call me now',  
          'hello call hello you tomorrow']
```

#### Step 3: Tokenization

```
In [18]: preprocessed_documents = [[w for w in d.split()]  
for d in sans_punctuation_documents]  
preprocessed_documents
```

```
Out[18]: [['hello', 'how', 'are', 'you'],  
          ['win', 'money', 'win', 'from', 'home'],  
          ['call', 'me', 'now'],  
          ['hello', 'call', 'hello', 'you', 'tomorrow']]
```

The notebook is running on a local host at localhost8888. The Windows taskbar at the bottom shows the time as 01:12 PM on 07-02-2021.

Tokenization

The screenshot shows a Jupyter Notebook interface with two sections. The first section, titled "Step 4: Count frequencies", contains two code cells. The first cell imports `pprint` and `Counter` from `collections`. The second cell creates a `frequency_list` by iterating over `preprocessed_documents` and printing the result. The output shows four `Counter` objects for different documents. The second section, titled "Implementing Bag of Words in scikit-learn", contains two code cells. The first cell imports `CountVectorizer` from `sklearn.feature_extraction.text`. The second cell creates a `count_vector` object. The output shows the `CountVectorizer` parameters.

```
In [19]: frequency_list = []
import pprint
from collections import Counter

In [20]: frequency_list = [Counter(d) for d in preprocessed_documents]
pprint.pprint(frequency_list)

[Counter({'hello': 1, 'how': 1, 'are': 1, 'you': 1}),
Counter({'win': 2, 'money': 1, 'from': 1, 'home': 1}),
Counter({'call': 1, 'me': 1, 'now': 1}),
Counter({'hello': 2, 'call': 1, 'you': 1, 'tomorrow': 1})]

Implementing Bag of Words in scikit-learn

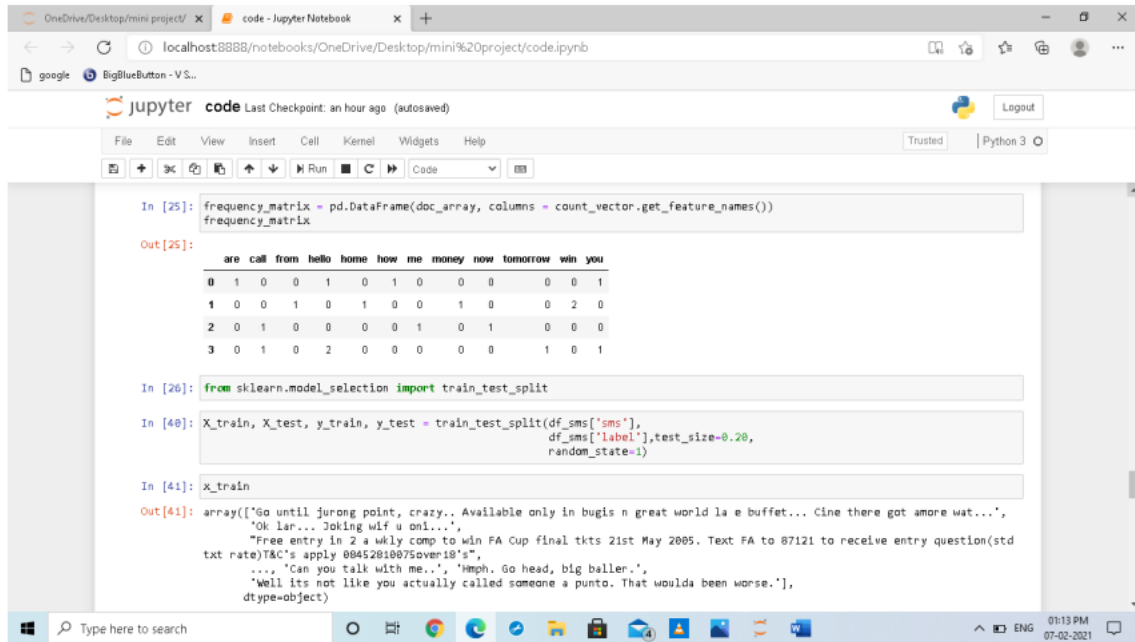
In [21]: from sklearn.feature_extraction.text import CountVectorizer
count_vector = CountVectorizer()

In [22]: count_vector
Out[22]: CountVectorizer(analyzer='word', binary=False, decode_error='strict',
dtype='class' 'numpy.int64', encoding='utf-8', input='content',
lowercase=True, max_df=1.0, max_features=None, min_df=1,
ngram_range=(1, 1), preprocessor=None, stop_words=None,
token_pattern='(?u)\\b\\w+\\b')
```

Counting frequencies and implementing bag of words in scikit-learn

The screenshot shows a Jupyter Notebook interface with a section titled "Data preprocessing with CountVectorizer()". It contains three code cells. The first cell calls `count_vector.fit(documents)` and `count_vector.get_feature_names()`. The output shows a list of feature names. The second cell calls `doc_array = count_vector.transform(documents).toarray()`. The output shows a 2D array of word counts. The third cell creates a `frequency_matrix` as a `pd.DataFrame` from the `doc_array` and the feature names.

Data Preprocessing with CountVectorizer()



```
In [25]: frequency_matrix = pd.DataFrame(doc_array, columns = count_vector.get_feature_names())
frequency_matrix

Out[25]:
```

	are	call	from	hello	home	how	me	money	now	tomorrow	win	you
0	1	0	0	1	0	1	0	0	0	0	0	1
1	0	0	1	0	1	0	0	1	0	0	2	0
2	0	1	0	0	0	0	1	0	1	0	0	0
3	0	1	0	2	0	0	0	0	0	1	0	1

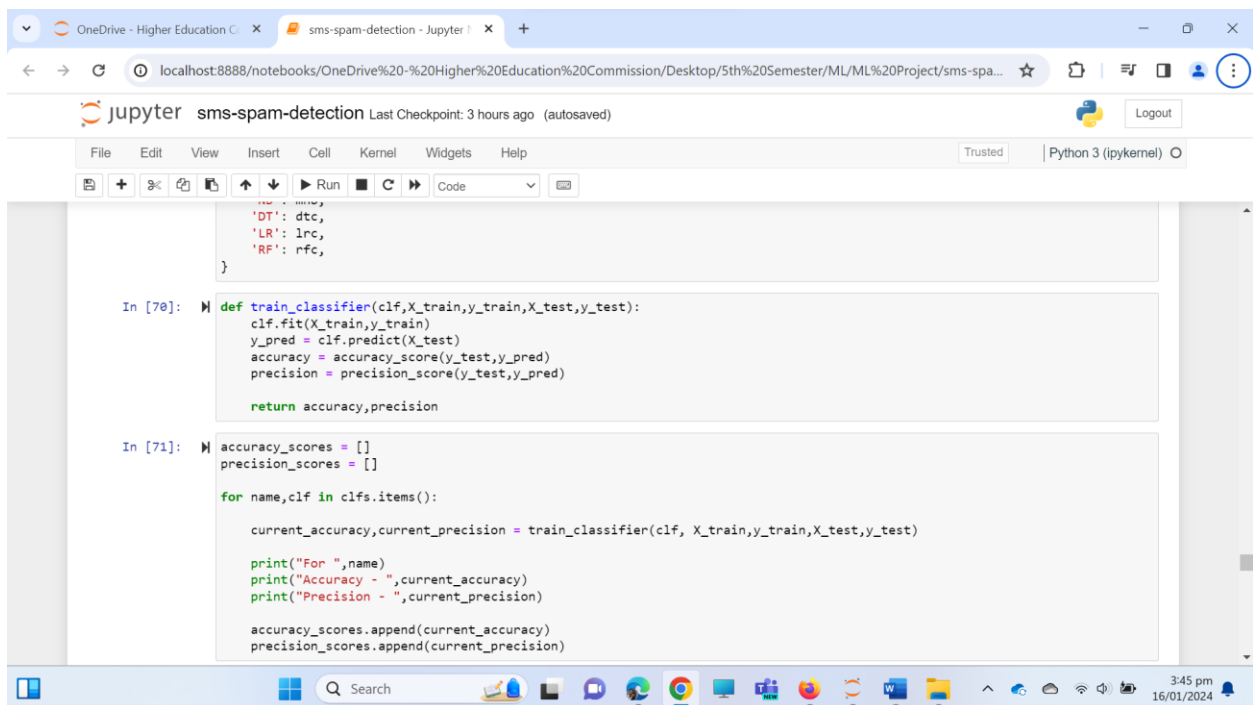
```
In [26]: from sklearn.model_selection import train_test_split

In [40]: X_train, X_test, y_train, y_test = train_test_split(df_sms['sms'],
df_sms['label'], test_size=0.20,
random_state=1)

In [41]: X_train

Out[41]: array(['Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...',
'Ok lar... Joking wif u oni...',
'Free entry in 2 a wkly comp to win FA Cup final tkts 21st May 2005. Text FA to 87121 to receive entry question(std
txt rate)T&C's apply 08452810075over18's',
..., 'Can you talk with me..', 'Hmph. Go head, big baller.',
'Well its not like you actually called someone a punto. That woulda been worse.'],
dtype=object)
```

## Splitting the data



```

'DT': dtc,
'LR': lrc,
'RF': rfc,
}

In [70]: def train_classifier(clf, X_train, y_train, X_test, y_test):
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)

return accuracy, precision

In [71]: accuracy_scores = []
precision_scores = []

for name, clf in clfs.items():

current_accuracy, current_precision = train_classifier(clf, X_train, y_train, X_test, y_test)

print("For ", name)
print("Accuracy - ", current_accuracy)
print("Precision - ", current_precision)

accuracy_scores.append(current_accuracy)
precision_scores.append(current_precision)
```

## Fitting Models on the Data

```
current_accuracy,current_precision = train_classifier(clf, X_train,y_train,X_test,y_test)

print("For ",name)
print("Accuracy - ",current_accuracy)
print("Precision - ",current_precision)

accuracy_scores.append(current_accuracy)
precision_scores.append(current_precision)

For KN
Accuracy - 0.9053140096618357
Precision - 1.0
For NB
Accuracy - 0.9603864734299516
Precision - 1.0
For DT
Accuracy - 0.9169082125603865
Precision - 0.8297872340425532
For LR
Accuracy - 0.9449275362318841
Precision - 0.941747572815534
For RF
Accuracy - 0.9652173913043478
Precision - 1.0

In [72]: performance_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy':accuracy_scores,'Precision':precision_scores}).sort_values(
```

## Model Evaluation Result

```
In [78]: temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_max_ft_3000':accuracy_scores,'Precision_max_ft_3000':precision_scores})

In [79]: temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_scaling':accuracy_scores,'Precision_scaling':precision_scores}).sort_values(

In [80]: new_df = performance_df.merge(temp_df,on='Algorithm')

In [81]: new_df_scaled = new_df.merge(temp_df,on='Algorithm')

In [82]: temp_df = pd.DataFrame({'Algorithm':clfs.keys(),'Accuracy_num_chars':accuracy_scores,'Precision_num_chars':precision_scores})

In [83]: new_df_scaled.merge(temp_df,on='Algorithm')

Out[83]:
```

	Algorithm	Accuracy	Precision	Accuracy_scaling_x	Precision_scaling_x	Accuracy_scaling_y	Precision_scaling_y	Accuracy_num_chars	Precision_num
0	KN	0.905314	1.000000	0.905314	1.000000	0.905314	1.000000	0.905314	1.0
1	NB	0.960386	1.000000	0.960386	1.000000	0.960386	1.000000	0.960386	1.0
2	RF	0.965217	1.000000	0.965217	1.000000	0.965217	1.000000	0.965217	1.0
3	LR	0.944928	0.941748	0.944928	0.941748	0.944928	0.941748	0.944928	0.9
4	DT	0.916908	0.829787	0.916908	0.829787	0.916908	0.829787	0.916908	0.8

OneDrive - Higher Education C... sms-spam-detection - Jupyter | x +

localhost:8888/notebooks/OneDrive%20-%20Higher%20Education%20Commission/Desktop/5th%20Semester/ML/ML%20Project/sms-spa... ☆

jupyter sms-spam-detection Last Checkpoint: 3 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

0	KN	0.905314	1.000000	0.905314	1.000000	0.905314	1.000000	0.905314	1.0
1	NB	0.960386	1.000000	0.960386	1.000000	0.960386	1.000000	0.960386	1.0
2	RF	0.965217	1.000000	0.965217	1.000000	0.965217	1.000000	0.965217	1.0
3	LR	0.944928	0.941748	0.944928	0.941748	0.944928	0.941748	0.944928	0.9
4	DT	0.916908	0.829787	0.916908	0.829787	0.916908	0.829787	0.916908	0.8

```
In [103]: from sklearn.ensemble import ExtraTreesClassifier
mnb = MultinomialNB()
etc = ExtraTreesClassifier(n_estimators=50, random_state=2)
from sklearn.ensemble import VotingClassifier

In [104]: voting = VotingClassifier(estimators=[('nb', mnb), ('et', etc)], voting='soft')

In [86]: voting.fit(X_train, y_train)

Out[86]:
VotingClassifier
  nb      et
  └─ MultinomialNB  ─ ExtraTreesClassifier
```

3:46 pm 16/01/2024

OneDrive - Higher Education C... sms-spam-detection - Jupyter | x +

localhost:8888/notebooks/OneDrive%20-%20Higher%20Education%20Commission/Desktop/5th%20Semester/ML/ML%20Project/sms-spa... ☆

jupyter sms-spam-detection Last Checkpoint: 3 hours ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3 (ipykernel)

```
print("Precision", precision_score(y_test, y_pred))

Accuracy 0.9671497584541063
Precision 1.0

In [88]: estimators=[('nb', mnb), ('et', etc)]
final_estimator=RandomForestClassifier()

In [89]: from sklearn.ensemble import StackingClassifier

In [90]: clf = StackingClassifier(estimators=estimators, final_estimator=final_estimator)

In [91]: clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)
print("Accuracy", accuracy_score(y_test, y_pred))
print("Precision", precision_score(y_test, y_pred))

Accuracy 0.9739130434782609
Precision 0.9618320610687023

In [92]: import pickle
pickle.dump(tfidf, open('vectorizer.pkl', 'wb'))
pickle.dump(mnb, open('model.pkl', 'wb'))

In [ ]:
```

3:47 pm 16/01/2024

## Models Improvements