# Building a Smarter AI Powered Spam Classifier

## Problem Definition

In machine learning, spam filtering protocols use instance-based or memory-based learning methods to identify and classify incoming spam emails based on their resemblance to stored training examples of spam emails Spam is any unsolicited communication sent in bulk.Usually sent via email ,spam is also distributed through text messages(SMS),social media,or phone calls. Spam messages often come in the form of harmless (though annoying)promotional emails .But sometimes spam is a fraudulent or malicious scam.

## Design Thinking

**1. The First Component to Consider When Building the AISolution Is the Problem Identification:**

Before developing a productor feature,it's essential to focus on the user's pain point and figure out the value proposition (value-prop) that users can get from your product.
A value proposition has to do with the value you promise to deliver to your customers should they choose to purchase your product.

**2. Have the Right Data and Clean It:**

Now,when you've framed the problem ,you need to pick the right data sources. It's more critical to get high-quality data than to spend time on improving the AImodel itself. Data falls under two categories

**3. Create Algorithms:**

When telling the computer what to do, you also need to choose how it will do it. That's where computer algorithms step in. Algorithms are mathematical instructions. It's necessary to create prediction or classification machine learning algorithms so, the AI model can learn from the dataset.

**4. Train the Algorithms**

Moving forward with how to create an AI,you need to train the algorithm using the collected data. It would be best to optimize the algorithm to achieve an AI model with high accuracy during the training process. How ever ,you may need additional data to improve the accuracy of your model.

**5. Opt for the Right Platform:**

Apart from the data required to train your AI model, you need to pick the right platform for your needs. You can go for an in-house or cloud framework. What's the main difference between these frameworks The cloud makes it easy for enterprises to experiment and grow as projects go into production and demand increases by allowing faster training and deployment of ML models.

- In-houseFrameworks

**6. Choose a Programming Language:**

There is more than one programming language ,including the classic C++,Java ,Python, and R. The latter two coding languages are more popular because they offer a robust set of tools such as extensive ML libraries. Make the right choice by considering your goals and needs.

## Algorithm

**Step1** : E-mail Data Collection.The data set contained in a corpus plays a crucial role in assessing the performance of any spam filter.

**Step2**:Pre-processing of E-mail content

**Step3**:Feature Extraction and Selection

**Step 4**: Implementation

**Step5**:Performance Analysis.

we load the dataset for the spam detection project.The dataset is stored in a CSV file located at'/content/spam.csv'. We use the pandas library to read the CSV file and do some preprocessing to the dataset like text Cleaning, Stemming and etc.

To perform natural language processing tasks,we'll first install the Natural Language Toolkit (NLTK) library.

**Analysis:**

We import the pandas library using import pandas as pd.
We use pd.read_csv() to read the CSV file containing the dataset.
The encoding='latin-1' argument is used to handle special characters.
We select only the relevant columns ('v1'forlabels,'v2'foremailcontent)using data[['v1', 'v2']].
Finally,we display the resulting DataFrame to inspect the loaded data.

**Code:**

```
Import pandas as pd

# Load the dataset
data=pd.read_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv',
encoding='latin-1')
data=data[['v1','v2']]

#Selecting only the relevant columns data

#printing
```

**Output**

| | v1 | v2 |
|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... |
| 1 | ham | Ok lar... Joking wif u oni... |
| 2 | spam | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | ham | U dun say so early hor... U c already then say... |
| 4 | ham | Nah I don't think he goes to usf, he lives aro... |
| ... | ... | ... |
| 5567 | spam | This is the 2nd time we have tried 2 contact u... |
| 5568 | ham | Will Ì_ b going to esplanade fr 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 × 2 columns

## Data Preprocessing

In this step,we perform datapreprocessing tasks,which include converting labels to binary values and removing duplicates from the dataset.

***Explanation:***

We use data['v1'].apply(lambdax:1ifx=='spam' else0) toconvertthelabels.'ham'is mapped to 0, and 'spam' is mapped to 1 in the 'v1' column.

We then remove duplicate rows from the dataset using data=data.drop_duplicates().

The resulting DataFrame is displayed to show the cleaned dataset.

**Code:**

```
#Convert'ham'to0and'spam'to1directlyinthe'v1'column data['v1'] =
data['v1'].apply(lambda x: 1 if x == 'spam' else 0)

# removing duplicates
data=data.drop_duplicates()
data
```

## Output

| | v1 | v2 |
|---|---|---|
| 0 | 0 | Go until jurong point, crazy.. Available only ... |
| 1 | 0 | Ok lar... Joking wif u oni... |
| 2 | 1 | Free entry in 2 a wkly comp to win FA Cup fina... |
| 3 | 0 | U dun say so early hor... U c already then say... |
| 4 | 0 | Nah I don't think he goes to usf, he lives aro... |
| ... | ... | ... |
| 5567 | 1 | This is the 2nd time we have tried 2 contact u... |
| 5568 | 0 | Will Ì_ b going to esplanade fr home? |
| 5569 | 0 | Pity, * was in mood for that. So...any other s... |
| 5570 | 0 | The guy did some bitching but I acted like i'd... |
| 5571 | 0 | Rofl. Its true to its name |

5169 rows × 2 columns

## Text Cleaning:

Text cleaning involves removing any unnecessary characters,symbols,or noise from the text data. This might include punctuation, special characters, and numbers.

**Explanation:**
We import the regular expression(re) module using import re.

The function clean_text() takes a string text as input and uses a regular expression to remove all characters except alphabetic character.
The cleaned text is then returned.

We apply this function to the 'v2' column of the DataFrame using data['v2'].apply(lambdax:clean_text(x)).Thiscleansthetextineachemail.

## Code:

```
import re
def clean_text(text):
    cleaned_text=re.sub(r'[^a-zA-Z]','',text) return
    cleaned_text

data['v2']= data['v2'].apply(lambdax: clean_text(x))
```

## Lower casing:

Converting all text to lowercase ensures that the model doesn't treat "Hello" and "hello" as different words.

### *Explanation:*

We use the str.lower() method to convert all text in the 'v2' column to lowercase. This helps standardize the text data and ensure that the model is not case-sensitive.

data['v2']=data['v2'].str.lower()

## Tokenization:

Tokenization involves splitting the text into individual words or tokens.The NLTK library can be used for this.

## Explanation:

In this code cell ,we use nltk. download('punkt') to download the necessary resources for tokenization from the Natural Language Toolkit (NLTK). This resource includes pre-trained models for tokenizing text into words or sentences. This step is essential for further text processing.

**Code:**

```
import nltk
nltk.download('punkt')
[nltk_data] Downloading package punkt
to/usr/share/nltk_data...
[nltk_data] Package punkt is already up-to-date!
```

**Output**

True

## Stemming:

Stemming reduce words to their base forms. This can help in reducing the dimensionality of the feature space.

**Explanation:**
We import the PorterStemmer class from the

NLTK library. We initialize an instance of the

PorterStemmer as stemmer.

We define a function stem_words(words) that takes a list of words and applies stemming to each word using the stemmer.stem() method.

We apply this function to the 'v2' column of the DataFrame, effectively reducing words to their base forms through stemming. This step can help improve the model's performance by reducing the feature space.

**Code:**

```
From nltk.stem import PorterStemmer
stemmer = PorterStemmer()


def stem_words(words):
    return[stemmer.stem(word) forword inwords]


data['v2']=data['v2'].apply(stem_word
s) data
```

## Output

| | v1 | v2 |
|---|---|---|
| 0 | 0 | [go, until, jurong, point, crazi, avail, onli,... |
| 1 | 0 | [ok, lar, joke, wif, u, oni] |
| 2 | 1 | [free, entri, in, a, wkli, comp, to, win, fa, ... |
| 3 | 0 | [u, dun, say, so, earli, hor, u, c, alreadi, t... |
| 4 | 0 | [nah, i, don, t, think, he, goe, to, usf, he, ... |
| ... | ... | ... |
| 5567 | 1 | [thi, is, the, nd, time, we, have, tri, contac... |
| 5568 | 0 | [will, b, go, to, esplanad, fr, home] |
| 5569 | 0 | [piti, wa, in, mood, for, that, so, ani, other... |
| 5570 | 0 | [the, guy, did, some, bitch, but, i, act, like... |
| 5571 | 0 | [rofl, it, true, to, it, name] |

5169 rows × 2 columns

## Preparations

**First we have to create the python environment for themodel to run , get dataset from**

"/kaggle/input/sms-spam-collection-dataset/spam.csv "

**Then clean and preprocess it, Import the necessary libraries in the Notebook and then load the dataset and run the head() function and thedrop() function to ignore the Nan and undefined columns .**

## CODE

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g.pd.read_csv)
```

```
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory
```

```
import os
for dirname, _, filenames in os.walk('/kaggle/input'):
for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when youcreate a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/,
but they won't be saved outside of the current session
```

## Importing necessary libraries

```
#necessary libraries

import numpy as np  # linear algebra import
pandas as pd
# data processing
import nltk from nltk.corpus
import stopwords from nltk.tokenize
import word_tokenize from nltk.stem
import PorterStemmer
```

## Loading the dataset

```
# Load the dataset

data = pd.read_csv('/kaggle/input/sms-spam-collection-dataset/spam.csv',
encoding='latin-1')
```

```
data.head()
```

**OUTPUT**

| | v1 | v2 | Unnamed: 2 | Unnamed: 3 | Unnamed: 4 |
|---|---|---|---|---|---|
| 0 | ham | Go until jurong point, crazy.. Available only ... | NaN | NaN | NaN |
| 1 | ham | Ok lar... Joking wif u oni... | 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 don't think he goes to usf, he lives aro... | NaN | NaN | NaN |

```
columns_to_drop = ['Unnamed: 2', 'Unnamed: 3',
'Unnamed: 4']
data = data.drop(columns_to_drop, axis=1, errors=
'ignore')
```

```
data.head()
```

## Tokenization and cleaning

Tokenization is the process of splitting the input and output texts into smaller units that can be processed by the LLM AI models. Tokens can be words , characters , subwords, or symbols, depending on the type and the size of the model

## CODE

```
# Clean the "v2" column
data['v2'] = data['v2'].str.lower()

# Tokenization and cleaning of data
def preprocess_text(text): tokens =
    word_tokenize(text)
    stop_words = set(stopwords.words('english')) filtered_tokens = [word for word in tokens
    if word.isa
lnum() and word n                    ot in stop_words]stemmer =
       PorterStemmer()
       stemmed_tokens = [stemmer.stem(word) for word in filtered_tokens]
return ' '.join(stemmed_tokens)
data['v2'] = data['v2'].apply(preprocess_text)
```

```
data.head()
```

**OUTPUT**

| | v1 | v2 |
|---|---|---|
| 0 | ham | go jurong point avail bugi n great world la e ... |
| 1 | ham | ok lar joke wif u oni |
| 2 | spam | free entri 2 wkli comp win fa cup final tkt 21... |
| 3 | ham | u dun say earli hor u c alreadi say |
| 4 | ham | nah think goe usf live around though |

```
from  sklearn.feature_extraction.text import TfidfVectorizer
from  sklearn.model_selection import train_test_split

# Load the dataset
tfidf_vectorizer = TfidfVectorizer()
tfidf_matrix = tfidf_vectorizer.fit_transform(data['v2']
)

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

# Split Data
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix,
data['v1'], test_size=0.2, random_state=42)

# Check the shape of the TF-IDF matrix and the split dataprint("TF-IDF Matrix
Shape:", tfidf_matrix.shape)
 print("Training Data Shape:", X_train.shape)
 print("Testing Data Shape:", X_test.shape)
```

**OUTPUT**

```
TF-IDF Matrix Shape: (5572, 8672)
Training Data Shape: (4457, 8672)
Testing Data Shape: (1115, 8672)
```

# Random Forest Classifier

Random forest is a commonly-used machine learning algorithm which combines the output of multiple decision trees to reach a singleresult.

## CODE

```
from  sklearn.ensemble import   RandomForestClassifier

from sklearn.metrics import accuracy_score, classification_report
 # Create a Random Forest classifier
 rf_classifier = RandomForestClassifier(random_state=42)

 # Train the classifier on the training data
 rf_classifier.fit(X_train, y_train)

 # Make predictions on the testing data
 y_pred = rf_classifier.predict(X_test)

 # Evaluate the model's performance
 accuracy = accuracy_score(y_test, y_pred) classification_rep =
 classification_report(y_test, y_pred)

 # Print the results
 print("Accuracy:", accuracy)
 print("classification report:\n", classification_rep)
```

## OUTPUT

```
Accuracy: 0.9766816143497757
classification report:
              precision    recall  f1-score   support

           0       0.97      1.00      0.99       965
           1       1.00      0.83      0.91       150

    accuracy                           0.98      1115
   macro avg       0.99      0.91      0.95      1115
weighted avg       0.98      0.98      0.98      1115
```

## Testing model

### Test1

```
input_text = """\apple Inc.Your iPhone 6 linked top***zm".edu) has
been used a few minutes
ago. To localize it,login now to your apple account ."""

# Apply the same preprocessing as in your previous code
input_text = input_text.lower()
# Add more preprocessing steps if needed

# Transform the input text into a TF-IDF vector
input_tfidf = tfidf_vectorizer.transform([input_text])

# Make a prediction using the trained Random Forest model
prediction = rf_classifier.predict(input_tfidf)
# predictions
if prediction[0] == 1:
  print("This message is predicted to be SPAM by trainedmodel.")
else:
    print("This message is predicted to be NOT SPAM by
trained model.")
```

### OUTPUT

This message is predicted to be NOT SPAM by trainedmodel.

**Test2**

```
input_text1 = "Hey, I'm mark. How are you?."
# Apply the same preprocessing as in your previous code
input_text1 = input_text1.lower()

# Transform the input text into a TF-IDF vector
input_tfidf = tfidf_vectorizer.transform([input_text1])

# Make a prediction using the trained Random Forest model
prediction = rf_classifier.predict(input_tfidf)

# perdictions
if prediction[0] == 1:
    print("This message is predicted to be SPAM by trained model.")
else:


print("This message isained          predicted      to be NOT SPAM by tr
model.")
```

**OUTPUT**

This message is predicted not spam

## Wordcloud of ham category

```
#generate wordcloud plot for not-spam messages
ham_wc=wc.generate(email_df[email_df["target"]==1]["transformed_message"].str
.cat(sep=" "))
plt.figure(figsize=(20,10))
plt.imshow(ham_wc)
plt.show()
```


```
#used words in spam messages
spam_corpus=list()
for msg in email_df[email_df['target']==0]["transformed_message"].to_list():
    for word in msg.split():
        spam_corpus.append(word)

len(spam_corpus)

OUTPUT
```

9883

```
#print the most common 50 words from the spam category messages
from collections import Counter
spam_top_50_common_words=pd.DataFrame(Counter(spam_corpus).most_common(50))
print(spam_top_50_common_words)

OUTPUT
```

```
         0    1
0     call  320
1     free  189
2        2  155
3      txt  141
4     text  122
5        u  119
6       ur  119
7    mobil  114
8     stop  104
9    repli  103
10   claim   98
11   prize   82
12       4   76
13     get   74
14     new   64
15  servic   64
16    tone   63
17    send   60
18  urgent   57
19   nokia   57
....
```

```
#used words in ham messages
ham_corpus=list()
for msg in email_df[email_df['target']==1]["transformed_message"].to_list():
    for word in msg.split():
        ham_corpus.append(word)
len(ham_corpus)
```

Out[24]:

34771

```
#most commnaly used 50 words from ham category messages
ham_top_50_common_words=pd.DataFrame(Counter(ham_corpus).most_common(
50))
print(ham_top_50_common_words)

OUTPUT
```

```
        0    1
0       u  871
1      go  401
2     get  349
3      gt  288
4      lt  287
5       2  284
6    come  272
7     got  236
8    like  234
9    know  234
10   call  232
11   time  217
12   good  212
13   want  208
14     ok  207
...
```

## Data Transformation

Using Count Vectorization

In [26]:

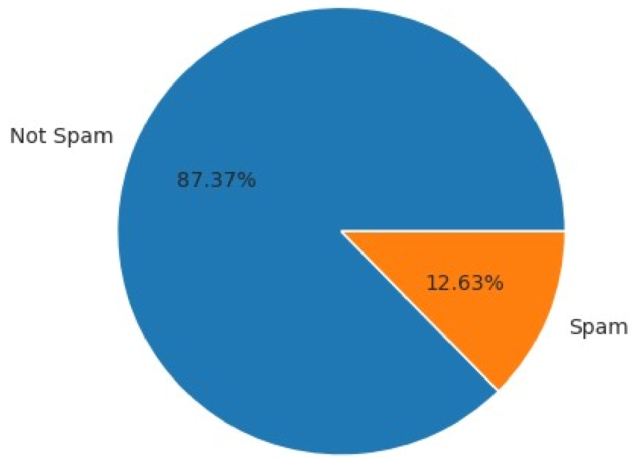
```
from sklearn.feature_extraction.text import CountVectorizer
cVector=CountVectorizer() #CountVectorizer is used to convert text into numeric array
x=cVector.fit_transform(email_df["transformed_message"]).toarray()
```

In [27]:

```
#seperating target column
y=email_df['target']
```

```
#check the distribution of target variable using Pie chart
plt.pie(y.value_counts().values,labels=["Not Spam","Spam"],autopct="%0.2f%%")
plt.show()
```

## OUTPUT




**Conclusion** : as we can see our dataset is imbalanced.

## Spliting data into Training and Testing sets into 80/20 ratio

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=43)
x_train.shape,y_train.shape,x_test.shape,y_test.shape
Out[29]:
((4135, 6629), (4135,), (1034, 6629), (1034,))
In [30]:

from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,confusion_matrix,classificatio
n_report

#function to evaluate the performance of model
def evaluate_model_performance(model,x_test,y_test):
    y_pred=model.predict(x_test)
    print("Accurary Score :
{}".format(np.round(accuracy_score(y_test,y_pred)*100,decimals=2)))
    print("Precision Score :
{}".format(np.round(precision_score(y_test,y_pred)*100,decimals=2)))
    print("Recall Score :
{}".format(np.round(recall_score(y_test,y_pred)*100,decimals=2)))
    print("F1 Score : {}".format(np.round(f1_score(y_test,y_pred)*100,decimals=2)))
    cm=confusion_matrix(y_test,y_pred)
    sns.heatmap(cm,fmt="d",annot=True,cmap="rainbow")
    plt.show()
    print("*Classification Report*********************************************")
    print(classification_report(y_test,y_pred))
```

In [31]:

```
#import models
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from xgboost import XGBClassifier
from sklearn.model_selection import cross_val_score
```

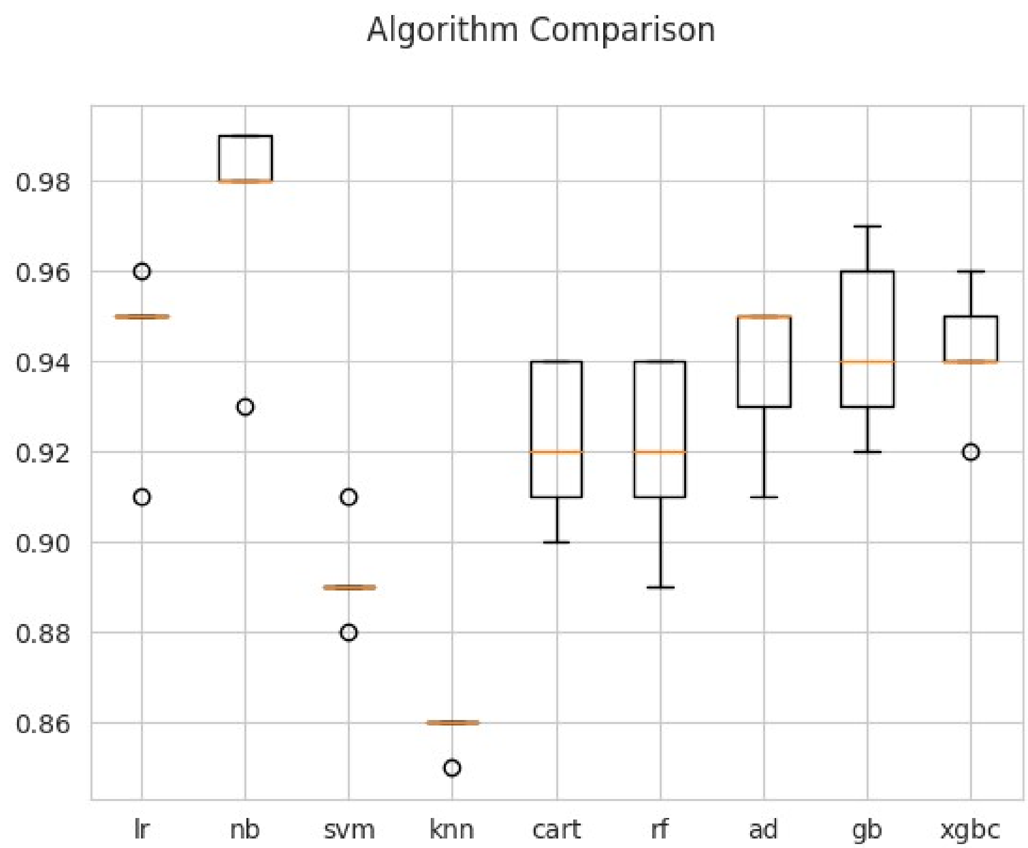
In [32]:

```
from sklearn.model_selection import StratifiedKFold
from imblearn.over_sampling import RandomOverSampler

# Define models
models = {
    "lr":LogisticRegression(),
    "nb":MultinomialNB(),
    "svm":SVC(),
    "knn":KNeighborsClassifier(),
    "cart":DecisionTreeClassifier(),
    "rf":RandomForestClassifier(),
    "ad":AdaBoostClassifier(),
    "gb":GradientBoostingClassifier(),
    "xgbc":XGBClassifier()
}

# Define oversampler for dealing with imbalance
oversampler = RandomOverSampler()

# Define cross-validation strategy for imbalanced data
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

model_scores=list()
# Loop through each model and evaluate its performance
for model_name, model in models.items():
    # Apply oversampling to training data
    X_resampled, y_resampled = oversampler.fit_resample(x, y)

    # Perform cross-validation
    scores = cross_val_score(model, X_resampled[:500], y_resampled[:500], cv=cv,
scoring="f1_micro")
    print(model_name," : ",np.round(np.mean(scores)*100,decimals=2))
    model_scores.append(scores)
# boxplot algorithm comparison
```

```
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(model_scores)
ax.set_xticklabels(models.keys())
plt.show()
```

## OUTPUT

```
lr  :  94.4
nb  :  97.4
svm  :  89.2
knn  :  85.8
cart  :  92.2
rf  :  92.0
ad  :  93.8
gb  :  94.4
xgbc  :  94.2
```




## CONCLUTION:

Thus the SMS Spam Collection dataset is preprocessed , transformed and trained based on the algorithm ,Overall the Ai powered spam classifier successfully classifies the dataset into ham and spam messages during the training of this model.